Rita Bonelli Luciano Pazzucconi Fabio Racchi

GUIDA AL COMMODORE PLUS 4

Un manuale completo per il PLUS 4 ripercorrendo C16 per te e C16 sempre di più

HOME E PERSONAL COMPUTER

GRUPPO EDITORIALE JACKSON

DIVISIONE LIBRI

Rita Bonelli   Luciano Pazzucconi   Fabio Racchi

GUIDA AL COMMODORE PLUS 4

Un manuale completo
per il PLUS 4
ripercorrendo C16 per te
e C16 sempre di più

GRUPPO
EDITORIALE
JACKSON
Via Rosellini, 12
20124 Milano

SOMMARIO

CAPITOLO 1: PANORAMICA DEL PLUS-4

1. 1 Introduzione 1
1. 2 La tastiera 2
1. 3 EDITOR 7
1. 4 Il video 9
1. 5 Il COMMODORE PLUS-4 come calcolatrice 11

CAPITOLO 2: PROGRAMMARE IN BASIC

2. 1 Il programma 17
2. 2 Il linguaggio BASIC 21
2. 3 Le istruzioni piu' elementari 23
2. 4 Esecuzione del programma 29
2. 5 Scrittura facilitata del programma ... 29
2 .6 Gestione del programma 31
2. 7 Strutture del programma 33
2 .8 Programmare la grafica 42
2. 9 Programmare l'animazione 53
2.10 Programmare il suono 55

CAPITOLO 3: OPERAZIONI AVANZATE IN BASIC

3. 1 Le variabili con indice 57
3 2 Il trattamento delle stringhe 59
3. 3 Lettura dati dall'interno del programma 63
3. 4 L'istruzione GET 64
3. 5 Le finestre sul video 65
3. 6 Definizione delle funzioni utente 66
3. 7 Alcune funzioni avanzate 67
3. 8 La gestione degli errori 68

CAPITOLO 4: COMUNICAZIONE CON LE PERIFERICHE

4. 1 Come il calcolatore comunica con le periferiche 73
4. 2 La tastiera vista come file 79
4. 3 Il video visto come file 80
4. 4 I file su cassetta 90
4. 5 I joystick 98

CAPITOLO 5: I FILE SU STAMPANTE

5. 1 Introduzione 103
5. 2 Istruzioni di stampa 105
5. 3 Modi di stampa e uso dei caratteri di controllo 110
5. 4 Gestione del buffer di stampa 126
5. 5 Come si compone uno stampato 128
5. 6 Copia del video testo su carta 132
5. 7 Copia del video grafico su carta 139

CAPITOLO 6: I FILE SU DISCO

6. 1 Introduzione 147
6. 2 Il DOS 152
6. 3 Gestione degli errori disco 162
6. 4 File di programma 166
6. 5 File SEQUENZIALI di dati 167
6. 6 File RANDOM di dati 177
6. 7 File RELATIVI di dati 200
6. 8 Programmi di utilita' 212

CAPITOLO 7: ARCHITETTURA DEL SISTEMA

7. 1 Introduzione........................... 223
7. 2 La memoria 224
7. 3 L'unita' centrale di elaborazione (CPU) 225
7. 4 I dispositivi di Ingresso/Uscita (I/O) 228
7. 5 La tastiera 228
7. 6 I joystick 231
7. 7 Il registratore a cassette 233
7. 8 Il video 237
7. 9 Il suono 242
7.10 La porta seriale 243
7.11 L'organizzazione della memoria 246

CAPITOLO 8: LA PROGRAMMAZIONE IN ASSEMBLER E IN LINGUAGGIO MACCHINA

8. 1 Introduzione 249
8. 2 Organizzazione della CPU 7501 252
8. 3 I modi di indirizzamento 258
8. 4 Il set di istruzioni 261
8.4.1 Le operazioni logiche 262
8.4.2 Istruzioni di trasferimento 264
8.4.3 Operazioni logico matematiche 266
8.4.4 Istruzioni di controllo del flusso 269

8.4.5 Istruzioni sui FLAG 271
8.4.6 Operazioni sullo STACK 272
8. 5 La gestione dell'interrupt 273
8. 6 Uso del comando MONITOR del BASIC 274

CAPITOLO 9: GRAFICA
9. 1 Introduzione 281
9. 2 Caratteri programmabili 281
9. 3 Copia dei caratteri da ROM 283
9. 4 Programmi per la creazione dei caratteri 286
9. 5 Caratteri a sfondo programmabile 296
9. 6 Caratteri multicolore 304
9. 7 Annullamento dello schermo 307
9. 8 Scorrimento fine (smooth scrolling) e registro di linea 308
9. 9 Riepilogo 316

CAPITOLO 10: TECNICHE DI PROGRAMMAZIONE E ESEMPI
10. 1 Introduzione 319
10. 2 Divisione parole in sillabe 319
10. 3 Disegno dei caratteri 322
10. 4 Contatori 324
10. 5 Grafico tridimensionale 327
10. 6 Utilizzo interrupt 332
10. 7 Esempio di USR 339

APPENDICE A: IL BASIC 3.5
A. 1 Introduzione 349
A. 2 Costanti e variabili 351
A. 3 Operatori aritmetici, relazionali e logici 354
A. 4 Comandi, istruzioni e funzioni 356

APPENDICE B: ISTRUZIONI ASSEMBLER 387

APPENDICE C: CODICI E NUMERI DEL CALCOLATORE 390

APPENDICE D: MESSAGGI DI ERRORE 417

APPENDICE E: UTILIZZO DELLA MEMORIA 423

APPENDICE F: MAPPA DELLA MEMORIA............ 431

APPENDICE G: REGISTRI DEL TED 439

APPENDICE H: VALORE DELLE NOTE 441

APPENDICE I: FUNZIONI MATEMATICHE
DERIVATE 445

APPENDICE L: CONVERSIONI DECIMALE,
ESADECIMALE, BINARIO 447

APPENDICE M: I 4 PROGRAMMI RESIDENTI
IN ROM ....

APPENDICE N: SCHEMA ELETTRICO 453

INDICE FIGURE E TABELLE

Fig. 1. 1	Tipi di carattere	9
Fig. 1. 2	Intervalli rappresentazione numeri	13
Fig. 2. 1	Schema della memoria	21
Fig. 2. 2	Diagramma a blocchi ES2.1	24
Fig. 2. 3	Immagine lettera A	43
Fig. 2. 4	Uso della memoria in modo grafico	45
Fig. 4. 1	File, record, campi	81
Fig. 4. 2	Diagramma a blocchi di FILESEQ ..	88
Fig. 4. 3	Fase aggiornamento di FILESEQ ...	93
Fig. 5. 1	Come i byte della pagina grafica danno il quadro video ...	140
Fig. 5. 2	Diagramma a blocchi sottoprogramma HARD4	144
Fig. 5. 3	Diagramma a blocchi sotto-programma interno a HARD4	145
Fig. 6. 1	Il dischetto nella sua busta	148
Fig. 6. 2	Schema non in scala del dischetto	149
Fig. 6. 3	Diagramma a blocchi dell'aggiornamento	171
Fig. 7. 1	Collegamento tra CPU e memoria ..	224
Fig. 7. 2	Ciclo di lettura da memoria	225
Fig. 7. 3	Ciclo di scrittura in memoria ...	226
Fig. 7. 4	Zoccolatura della CPU 7501	226
Fig. 7. 5	Segnale Φ0, il clock del sistema	228
Fig. 7. 6	Organizzazione della tastiera ...	229
Fig. 7. 7a	Collegamento con i Joystick	232
Fig. 7. 7b	Schema elettrico dei Joystick ...	232
Fig. 7. 8a	Collegamento della testina all'interno del registratore durante la fase RECORD	235
Fig. 7. 8b	Livelli di tensione e magnetizzazione della testina del registratore	236
Fig. 7. 9	Collegamento della testina in fase PLAY	236

Fig. 7.10 Segnale video per pilotare un monitor 238
Fig. 7.11a Mappa della memoria dei caratteri 240
Fig. 7.11b Mappa della memoria del colore e della luminosita' dei caratteri 240
Fig. 7.12 Connessioni della presa AUDIO/VIDEO 241
Fig. 7.13 Registri per la gestione del suono 242
Fig. 7.14 Schema della porta di I/O seriale (SERIAL BUS) 244
Fig. 7.15 Schema elettrico del SERIAL BUS 246
Fig. 7.16 Organizzazione della memoria 247
Fig. 8. 1 Esadecimale, binario e decimale 250
Fig. 8. 2 Analisi del numero 101 251
Fig. 8. 3 Architettura interna della CPU 7501 253
Fig. 8. 4 Registri della CPU 7501 a disposizione del programmatore 255
Fig. 8. 5 STACK e STACK POINTER 258
Fig. 8. 6 Formato dei risultati del sottoprogramma LEGGIJOYASS 277
TAB. 9. 1 Combinazione di tasti per ottenere i 4 colori di sfondo 302
Fig. 9. 1 Diagramma a blocchi sottoprogramma GRAF16 313
Fig. 10. 1 Rappresentazione in assonometria 328
Fig. 10. 2 Grafico della funzione Z3=SIN(X3*Y3) 330
Fig. 10. 3 Grafico della funzione Z3=SIN(Y3*2) 331
Fig. 10. 4 Grafico della funzione Z3=SIN(X3*X3+Y3*Y3) 331
Fig. 10. 5 Grafico della funzione Z3=(X3*X3+Y3*Y3)*(X3*X3+Y3*Y3)/32-1 . 332
Fig. 10. 6 Tre posizioni dell'aeroplanino 333
Fig. 10. 7 Diagramma a blocchi di INT1 337
Fig. 10. 8 Diagramma a blocchi di INT2 338
Fig. 10. 9 Algoritmo di ordinamento 342
Fig. 10.10 Schema a blocchi del programma ESEMPIO DI USR 345
Fig. 10.11 Schema a blocchi sottoprogramma . 346

Fig. A. 1 Parametri CIRCLE 360
Fig. A. 2 Schemi logici FOR e DO...LOOP ... 367
Fig. A. 3 Schema chiamata SOTTOPROGRAMMA .. 369
Fig. A. 4 Schema logico IF...THEN...ELSE .. 371
Fig. C. 1 Indirizzi di memoria 390
Fig. C. 2 Disposizione dei BIT in un BYTE . 390
Fig. C. 3 Pesi dei BIT in un BYTE 391
Tab. C. 1 Caratteri, codice ASCII
e D/CODE 400
Tab. C. 2 Parole chiave, TOKENS e
abbreviazioni 404
Tab. C. 3 TOKENS in ordine di codice 405
Tab. C. 4 Parole riservate codificate
ASCII 406
Tab. C. 5 Codici di controllo 407
Tab. C. 6 Relazione tra codice ASCII
e D/CODE 409
Fig. C. 4 Descrizione dei caratteri 410
Fig. E. 1 Utilizzo della memoria
e puntatori 424
Tab. L. 1 Conversioni dec. esadec. 447

Gli autori ringraziano James Bachmann, Amministratore Delegato, e Sergio Messa, Direttore Generale, della Commodore Italiana s.p.a., per aver messo a disposizione le apparecchiature e la documentazione necessarie alla realizzazione dell'opera.

Nel testo per esigenze tipografiche la FRECCIA VERSO L'ALTO, che ha il significato di elevamento a potenza, e' stampata col carattere ^.
I numeri esadecimali compaiono con il suffisso "H" o con il prefisso "$", nei punti dove la mancanza dei medesimi potrebbe generare confusione.
Nei programmi alcune volte si fa uso di caratteri di controllo all'interno delle virgolette; riportiamo l'elenco dei caratteri di controllo e il loro significato nei due set maiuscolo/grafico e minuscolo/maiuscolo.

maiuscolo/grafico	minuscolo/maiuscolo
HOME	home
CLR HOME	clr home
CURSORE SU	cursore su
CURSORE GIU'	cursore giu'
CURSORE A SINISTRA	cursore a sinistra
CURSORE A DESTRA	cursore a destra
CURSORE NERO	cursore nero
CURSORE BIANCO	cursore bianco
CURSORE ROSSO	cursore rosso
CURSORE CIANO	cursore ciano
CURSORE VIOLA	cursore viola
CURSORE VERDE	cursore verde
CURSORE BLU	cursore blu
CURSORE GIALLO	cursore giallo
CURSORE ARANCIO	cursore arancio
CURSORE MARRONE	cursore marrone
CURSORE GIALLO-VERDE	cursore giallo-verde
CURSORE ROSA	cursore rosa
CURSORE BLU-VERDE	cursore blu-verde
CURSORE BLU CHIARO	cursore blu chiaro
CURSORE BLU SCURO	cursore blu scuro
CURSORE VERDE CHIARO	cursore verde chiaro
REVERSE ON	reverse on
REVERSE OFF	reverse off
FLASH ON	flash on
FLASH OFF	flash off

PREFAZIONE

Questo libro vuole essere una guida alla conoscenza del COMMODORE PLUS-4 e all'apprendimento della programmazione in linguaggio BASIC e in linguaggio ASSEMBLER.
Viene presentata l'implementazione del BASIC 3.5 disponibile sul COMMODORE PLUS-4. Inoltre, e' illustrato il DOS residente sull'unita' a floppy disk 1541 e viene insegnato l'uso completo della stampante grafica MPS-803.
Lo studio dell'ASSEMBLER viene sviluppato appoggiandosi alle possibilita' offerte dal comando MONITOR del BASIC, e viene affrontato anche il problema del linguaggio macchina della CPU 7501 e della sua struttura hardware.
Le spiegazioni teoriche sono sempre accompagnate da esempi commentati, e per questo il libro puo' essere molto utile anche ai principianti.
Particolare attenzione viene dedicata all'uso delle periferiche, compresa la gestione dei file su disco. La grafica e' uno degli argomenti piu' approfonditi, collegandola alle possibilita' del processore video.

Nei primi 3 capitoli si ha una panoramica delle possibilita' del COMMODORE PLUS-4 e delle problematiche relative alla programmazione, affrontando gli argomenti in modo rigoroso, ma a livello elementare. Nei capitoli seguenti gli argomenti vengono approfonditi, ma si cerca di rendere le spiegazioni abbastanza semplici.

Il Capitolo 4 tratta le periferiche tastiera, video, cassetta e joystick. Il Capitolo 5 presenta completamente la stampante grafica MPS-803 e riporta diversi programmi per ottenere la copia del video testo e del video grafico. Il Capitolo 6 tratta i file su disco, riportando tutte le informazioni necessarie per usare bene il DOS; inoltre sono presenti programmi commentati per gestire archivi di dati di tipo sequenziale, e di tipo random, random-user e relativo, anche con indice. Il Capitolo 7 si sofferma sull'architettura del sistema ponendo l'accento sulla parte hardware e mettendo in grado anche gli inesperti di comprendere come e' costruito il calcolatore. Il Capitolo 8 tratta del linguaggio macchina e dell'uso completo del comando MONITOR. Il Capitolo 9 approfondisce l'argomento della grafica, utilizzando anche il linguaggio macchina. Nel Capitolo 10 sono presenti alcuni programmi commentati.

Completano il volume le appendici, che riportano: la scheda completa del BASIC 3.5, le istruzioni ASSEMBLER, i codici e i numeri del sistema, i messaggi d'errore, la mappa e l'utilizzo della memoria, l'elenco dei registri di I/O, il valore delle note, le funzioni matematiche derivate, le conversioni tra i sistemi di numerazione, una breve sintesi dei 4 programmi residenti in ROM e lo schema elettrico del calcolatore.

Il libro contiene molti programmi commentati, sia in BASIC che in ASSEMBLER, che risultano molto utili, dopo attenta lettura, per imparare a programmare. Questi, inoltre, possono essere usati dal lettore, sia nella versione presentata, che apportando alcune modifiche, o adoperando le routine piu' significative, per il proprio lavoro.

CAPITOLO 1

PANORAMICA DEL PLUS - 4

1.1 INTRODUZIONE

In questo libro parliamo del COMMODORE PLUS-4; esso e' un calcolatore della famiglia dei personal, cioe' un calcolatore di modesto costo e dimensioni, ma piuttosto potente.

Le principali parti componenti il nostro calcolatore sono:
. unita' centrale, cioe' il calcolatore,
. unita' di ingresso principale, la tastiera,
. unita' di uscita principale, il video.

L'unita' centrale comprende il microprocessore base del calcolatore, chiamato anche CPU (Central Processing Unit), e tutta l'elettronica necessaria al funzionamento delle periferiche, tastiera e video. Tra l'altro esso comprende le parti necessarie per il collegamento del registratore, periferica esterna per l'ingresso e l'uscita dei dati su cassetta magnetica.

Il calcolatore e le periferiche sono l'HARDWARE, la parte fisica delle apparecchiature. Il calcolatore non funziona se non si dispone anche del SOFTWARE, cioe' dei programmi. I programmi, registrati su un supporto magnetico come le cassette, si possono comprare e si possono preparare da soli, dopo aver imparato.

Il calcolatore elettronico e' una macchina che funziona eseguendo programmi. I divertenti videogiochi sono dei programmi.

Una parte importante dello hardware del calcolatore e' la MEMORIA, cioe' quella parte che permette di conservare programmi e dati e quindi di lavorare.
Nel COMMODORE PLUS-4 una parte di memoria e' stata preventivamente trattata e contiene dei programmi registrati in modo indelebile; essa viene chiamata memoria ROM (Read Only Memory).
Il SOFTWARE residente comprende:

.il SISTEMA OPERATIVO, che comprende tutti i programmi necessari per gestire il calcolatore;
.l'interprete BASIC, che consente di scrivere ed eseguire programmi in linguaggio BASIC;
.un programma di elaborazione testi (word-processor).
.un programma di gestione dati (data-base).
.una tabella elettronica (spread-sheet).
.un programma di visualizzazione grafica.

La parte di memoria dove puoi scrivere e leggere si chiama RAM.
Il calcolatore viene quasi sempre collegato ad altre apparecchiature; si chiamano INTERFACCE i dispositivi che consentono di collegare tra loro componenti diverse.

Non ci preoccupiamo di descrivere i collegamenti tra il calcolatore e le sue periferiche standard, dal momento che queste informazioni sono contenute nei manualetti allegati alle apparecchiature e qui facciamo riferimento al sistema pronto per lavorare.

1.2 LA TASTIERA

Il tuo COMMODORE PLUS-4 ha una tastiera composta di 67 tasti: 63 "normali" e 4 "di funzione". I 4 tasti "di funzione" sono in realta' tasti a cui tu puoi assegnare il significato che vuoi (ma questo lo vedremo bene nel seguito); i 63 "normali" sono tasti che hanno ciascuno un significato, che tu non puoi cambiare, e che sono usati principalmente per introdurre nel calcolatore i dati relativi ai programmi, ai numeri, ai comandi, ecc.
Poiche' vi sono nella tastiera 3 tasti SHIFT (2 tasti SHIFT e un tasto SHIFT LOCK), e 2 tasti CONTROL, in realta' vi sono 60 funzioni diverse che sono svolte da 63 tasti.

TASTI ALFABETICI E NUMERICI

I tasti della tastiera possono essere divisi in diversi tipi, a seconda della funzione che svolgono; i tasti con i numeri, ad esempio, li possiamo chiamare "numerici", e quelli con le lettere dell'alfabeto, "alfabetici". Possiamo chiamare "alfanumerici" l'insieme dei tasti numerici e alfabetici, piu' i segni di punteggiatura, spazio, parentesi, #, dollaro, apici, ^, ecc.

TASTI SHIFT, CBM LOGO e CTRL

Vi sono dei tasti che da soli non svolgono nessuna funzione, e che vanno premuti insieme ad altri, per cambiarne il signi-

ficato; essi sono SHIFT, CBM LOGO (il tasto in basso a sinistra, sotto RUN/STOP), e CTRL. Possiamo considerare i precedenti come "tasti di servizio".

Il COMMODORE PLUS-4 dispone di 2 SET di caratteri in alternativa tra loro, cioe' ne puo' essere attivo uno solo per volta. Al momento dell'accensione e' attivo il SET MAIUSCOLO/GRAFICO, nel quale le lettere compaiono in maiuscolo e si possono ottenere i caratteri grafici con l'uso di SHIFT (quelli posti a destra sul tasto) e di CBM LOGO (quelli posti a sinistra sul tasto). L'altro SET si chiama MINUSCOLO/MAIUSCOLO; quando esso e' attivo le lettere compaiono in minuscolo. Per ottenere le maiuscole si deve usare lo SHIFT, i caratteri grafici che ottenevi con SHIFT nell'altro set non sono piu' disponibili, mentre si ottengono con il tasto CBM LOGO quelli posti a sinistra sul tasto.
Premendo insieme CBM LOGO e SHIFT si passa dal set maiuscolo/grafico a quello minuscolo/maiuscolo, e viceversa.

Il tasto CBM LOGO usato insieme ai tasti numerici da 1 a 8, fa cambiare il colore del cursore nel colore stampato in basso sul tasto premuto.
Usato con gli altri tasti, CBM LOGO produce gli stessi effetti che lo SHIFT.
Il tasto CBM LOGO ha un'altra importante funzione: usato mentre il calcolatore sta stampando sul video, rallenta notevolmente l'operazione di SCROLLING, in modo da permettere la lettura delle scritte che scorrono.
Il tasto CTRL (control) va premuto, come SHIFT e CBM LOGO, insieme al tasto con cui si vuole usare. Principalmente con CTRL vanno premuti i tasti numerici da 1 a 8, per cambiare il colore del cursore nel colore stampato in alto sul tasto premuto, o i tasti 9 e 0, per inserire e disinserire l'effetto REVERSE. CTRL puo' essere usato anche in unione ai tasti , (virgola) e . (punto) per inserire e disinserire l'effetto FLASH.

Ecco la corrispondenza tasti/colori in italiano:

Tasto	con CBM LOGO	con CTRL
1	ORNG = ARANCIONE	BLK = NERO
2	BRN = MARRONE	WHT = BIANCO
3	YL GRN = GIALLO-VERDE	RED = ROSSO
4	PINK = ROSA	CYN = CIANO
5	BL GRN = VERDE-BLU	PUR = PORPORA
6	L BLU = BLU CHIARO	GRN = VERDE
7	D BLU = BLU SCURO	BLU = BLU
8	L GRN = VERDE CHIARO	YEL = GIALLO

TASTI DI CONTROLLO

Alcuni tasti non stampano un carattere, ma svolgono una particolare funzione di controllo del cursore; li possiamo chiamare tasti di "controllo". Appartengono a questa categoria i tasti RETURN, CLR/HOME, INST/DEL, RUN/STOP, e i tasti che spostano il cursore lungo lo schermo (i 4 con le frecce), e ESC.

TASTO RETURN
Il tasto RETURN serve per terminare la linea corrente, e memorizzarla. Se la linea non inizia con un numero, il suo contenuto viene subito interpretato come comando BASIC e mandato in esecuzione. Se invece inizia con un numero, essa viene memorizzata nell'area di memoria riservata al programma. Questo tasto puo essere "pericoloso", in quanto, premuto inavvertitamente, puo' far entrare nel programma linee indesiderate, o cancellare linee che gia' esistono; e' quindi buona norma non premere RETURN in continuazione, sopra uno schermo pieno di scritte, ma conviene piuttosto usare i tasti di movimento cursore e scendere dove lo schermo e' pulito. Se non vuoi che la linea corrente venga interpretata, puoi premere SHIFT-RETURN; questa operazione ti permette di terminare la linea corrente senza ne' memorizzarla, ne' mandarla in esecuzione: essa resta sul video.

TASTO CLEAR/HOME
Il tasto CLEAR/HOME ha due significati; usato da solo porta il cursore alla posizione HOME, cioe' nell'angolo in alto a sinistra, usato insieme allo SHIFT cancella tutto lo schermo, e porta il cursore nell'angolo in alto a sinistra.

TASTO INST/DEL
Il tasto INST/DEL, come il tasto CLEAR/HOME, ha due significati. Usato da solo svolge la funzione DELETE, cioe' cancella il carattere a sinistra del cursore, spostando a sinistra tutti gli eventuali caratteri presenti, sulla stessa linea, a destra del cursore. Usato insieme allo SHIFT invece significa INSERT, e inserisce uno spazio tra il carattere che precede il cursore e quello sotto il cursore. Ovviamente i caratteri che si trovano a destra del cursore sono automaticamente spostati a destra, per fare posto allo spazio appena inserito. Per tutti gli spazi inseriti viene automaticamente attivato il modo "INSERT", che ti permette di inserire caratteri di controllo senza mandarli subito in esecuzione (vedi Paragrafo 1.3)

TASTI DI MOVIMENTO CURSORE
Sono i tasti che si trovano in basso a destra sulla tastiera. La loro funzione e' quella di spostare il cursore lungo lo schermo, senza cancellare quello che vi si trova sotto. L'EDITOR del

COMMODORE PLUS-4 (programma del sistema che facilita il colloquio video/tastiera) e' molto potente, e ti permette di correggere le linee semplicemente passandovi sopra con il cursore, effettuando le correzioni, e premendo RETURN alla fine.

TASTO RUN/STOP

Questo tasto ha tre diverse funzioni:

. Premuto da solo, durante l'esecuzione di un programma, lo arresta, e fa si' che venga emesso il messaggio BREAK IN ... (numero linea dell'arresto). Il programma puo' riprendere con l'istruzione CONT (continua). Se dopo l'arresto del programma e' stata introdotta qualche nuova linea, o eseguita un'istruzione CLR, allora la ripresa del programma e' impossibile, e viene emesso il messaggio CAN'T CONTINUE.

. Premuto insieme a SHIFT fa si' che venga caricato il primo programma da disco, e vada in esecuzione automaticamente.

. Premuto insieme a RESET arresta il programma e fa entrare in MONITOR (vedi Paragrafo 8.6).

TASTO ESC (ESCAPE)

Per ultimo descriviamo il tasto ESC (escape). Svolge delle funzioni molto potenti, e puo' tornare utile imparare almeno quelle piu' importanti, per risparmiare tempo in fase di scrittura programmi.

A differenza di SHIFT, CBM LOGO, e CTRL, questo tasto non deve essere premuto insieme al tasto con cui va usato, ma va premuto prima. Ecco la lista dei tasti che possono essere premuti dopo ESC, e l'effetto che producono:

A: Inserimento automatico; quando si scrive su una riga, il carattere che si trova sotto il cursore viene spostato a destra, anziche' essere cancellato.
C: Cancella inserimento automatico.
D: Cancella tutta la linea dove si trova il cursore.
I: Inserisce una linea.
J: Porta il cursore all'inizio della linea corrente.
K: Porta il cursore alla fine della linea corrente.
L: Abilita lo "scrolling", movimento automatico delle scritte verso l'alto, per liberare l'ultima linea in basso.
M: Disabilita lo "scrolling".
N: Riporta lo schermo a 25 righe, 40 colonne.
O: Cancella il modo QUOTE e INSERT. Nel modo QUOTE si entra dopo aver aperto le virgolette; come si entra nel modo INSERT e' gia' stato spiegato. In questi modi i caratteri di controllo cursore e colore sono stampati come caratteri in campo inverso, anziche' produrre subito il loro effetto. ESC O cancella l'effetto FLASH e REVERSE (vedi Paragrafo 1.3).
P: Cancella la linea corrente dall'inizio al cursore.
Q: Cancella la linea corrente dal cursore alla fine.

R: Riduce lo schermo a 23 righe, 38 colonne. Puo' servire per quei televisori che non riescono a visualizzare tutta la grandezza dello schermo.
T: Posiziona l'angolo sinistro in alto della finestra video (vedi Paragrafo 3.5).
B: Posiziona l'angolo in basso a destra per la finestra video (vedi Paragrafo 3.5).
V: Fa scorrere le scritte dello schermo (scrolling) di una riga verso l'alto.
W: Fa scorrere le scritte dello schermo di una riga verso il basso.
X: Non produce alcun effetto; premi X se hai premuto ESC inavvertitamente, e non vuoi eseguire nessuna delle funzioni di ESCAPE.
E' possibile ottenere le funzioni di escape anche da programma, mediante l'istruzione PRINT CHR$(27). Ad esempio, per ottenere l'inserimento automatico:
PRINT CHR$(27)"A".

TASTI DI FUNZIONE

Hai notato sicuramente i 4 tasti in alto, con le scritte f1, f2, ... f7, HELP.
Questi tasti sono molto comodi, poiche' ad essi puo' essere assegnato il significato che vuoi. Per sapere quale significato hanno ora questi tasti, puoi digitare l'istruzione:
KEY
e premere RETURN.

Il calcolatore scrivera' sul video i significati attuali dei tasti funzione.
Se invece vuoi assegnare al tasto f1 una determinata funzione, puoi digitare:
KEY 1,"COMMODORE PLUS-4"
e poi premere RETURN.
In questo modo, quando premi f1 appare sul video la scritta COMMODORE PLUS-4. Ovviamente, per far tornare i tasti alle loro funzioni originali, puoi assegnare nuovamente a ciascun tasto la funzione precedente, o premere il tasto di RESET, che si trova vicino all'interruttore di alimentazione.

TASTO RESET

Il tasto RESET e' l'unico che non si trova sulla tastiera; esso infatti e' situato vicino all'interruttore di alimentazione, sul lato destro del tuo COMMODORE PLUS-4. Premere questo tasto assomiglia un po' a spegnere e riaccendere il calcolatore, ma con alcune differenze:
. Il contenuto della memoria utente non viene alterato, anche se

i puntatori del BASIC vengono riportati nella condizione iniziale.

. Tenendo premuto RUN/STOP e premendo RESET si arresta il programma corrente, anche se una imprudente routine di TRAP (vedi Paragrafo 3.8) lo impedisce normalmente. Questa operazione ti porta in MONITOR; per tornare al BASIC basta premere X e RETURN. Il programma e' ancora perfettamente in memoria, e anche il valore delle variabili e' conservato.

. Fai attenzione a non premere RUN/STOP mentre stai accendendo il calcolatore, altrimenti entri in MONITOR, ma uscendo da esso, con X, il BASIC non e' in grado di funzionare, poiche' in questo modo e' stata saltata la routine di inizializzazione del BASIC.

1.3 EDITOR

Viene chiamato EDITOR quella parte di SISTEMA OPERATIVO che provvede a gestire il colloquio tra la tastiera e il video. L'EDITOR del tuo COMMODORE PLUS-4 ha delle particolarita' che e' bene mettere a fuoco.

Innanzi tutto e' un EDITOR FULL SCREEN. Questo vuol dire che quando premi RETURN viene considerata valida tutta la linea dove si trova il cursore dal precedente RETURN all'attuale RETURN, e non solo i caratteri premuti ex novo. Questa particolarita' aiuta molto durante la fase di correzione dei programmi: non occorre digitare tutta la linea sbagliata, ma basta correggere solo dove necessario, e premere RETURN.

Inoltre accetta delle linee lunghe fino a 88 caratteri, poiche' considera linee logiche anziche' linee fisiche. Il fatto che linee piu' lunghe di 88 caratteri diano il messaggio STRING TOO LONG e' dovuto alla dimensione limitata della zona di memoria dedicata ad accettare i caratteri in ingresso dalla tastiera, e non a limiti imposti dall'EDITOR.

LINEE LOGICHE E LINEE FISICHE

Sul video del tuo televisore, appena accendi il calcolatore, appare uno schermo dove puoi scrivere dei caratteri, digitandoli dalla tastiera. Se conti i caratteri che puoi porre su una riga, vedi che sono 40; se poi conti quante righe e' possibile scrivere sullo schermo, vedi che sono 25.

Le righe che abbiamo appena contato, le chiamiamo LINEE FISICHE; esse infatti sono fisicamente 25, e il loro numero non puo' variare, ne' la loro lunghezza puo' superare i 40 caratteri. Abbiamo accennato che il calcolatore accetta in ingresso linee fino a un massimo di 88 caratteri. E' lecito domandarsi come sia possibile creare una linea di 88 caratteri su uno schermo che possiede solamente 40 colonne. La spiegazione e' che le linee in

questione non sono linee fisiche (tali linee infatti contengono sempre 40 caratteri, di cui alcuni eventualmente sono spazi bianchi), ma LINEE LOGICHE. Il concetto di linea logica e' molto semplice; una LINEA LOGICA e' un insieme di una o piu' linee fisiche. Una linea logica inizia dopo la pressione del tasto RETURN, e continua finche' non si preme nuovamente RETURN, a patto di non usare i tasti di controllo del cursore e CLEAR/HOME. Se viene premuto un tasto dopo il quarantesimo della linea corrente, l'EDITOR porta il cursore nella nuova linea, e "collega" la nuova linea a quella precedente. Lo stesso avviene dopo l'ottantesimo carattere. Esiste nell'area di memoria riservata al SISTEMA OPERATIVO una "mappa", che tiene conto delle linee fisiche che sono l'inizio di una linea logica, e di quelle che invece sono il proseguimento della linea logica precedente.

I "MODI" DELL'EDITOR: "QUOTE" E "INSERT"

Se hai gia' avuto modo di scrivere qualche programma sul COMMODORE PLUS-4, hai notato probabilmente che accade qualcosa di strano quando premi i tasti SHIFT-2 (cioe' apri le virgolette); alcuni dei caratteri di controllo vengono stampati in maniera strana, anziche' essere immediatamente eseguiti. Avviene che l'EDITOR entra nel modo QUOTE (modo virgolette). Quando l'EDITOR si trova in tale stato, i caratteri di controllo, escluso RETURN, vengono stampati come particolari caratteri in campo inverso, dentro le virgolette. Questi caratteri di controllo verranno eseguiti quando la STRINGA (insieme di caratteri) sara' stampata. Come esempio digita:

A$="SHIFT-CLEAR/HOME"

e premi RETURN.

Il tasto SHIFT-CLEAR/HOME appare come un cuoricino in campo inverso. Digita poi:

PRINT A$

e premi RETURN.

Vedi che tutto lo schermo e' stato cancellato, e il cursore si trova nell'angolo in alto a sinistra. E' stato "stampato" il carattere di controllo CLEAR (cancella lo schermo) e lo schermo e' stato cancellato. L'utilita' dell'uso dei caratteri di controllo all'interno di stringhe da stampare e' evidente. Puoi controllare da programma la posizione del cursore, ed eseguire i caratteri di controllo come FLASH ON/OFF, REVERSE ON/OFF, i caratteri per cambiare il colore del cursore, ecc.

Per uscire dal modo QUOTE esistono diversi modi:

. Premere RETURN e terminare la linea.

. Premere SHIFT-RETURN e portare il cursore alla nuova linea, senza introdurre la linea corrente.

. Chiudere le virgolette.

. Premere ESC O (non zero, ma O come Otranto. Vedi Paragrafo 1.2).
Il modo INSERT (modo inserimento) e' fondamentalmente identico al QUOTE, ma diverso e' il modo di entrarvi; si entra in INSERT premendo i tasti SHIFT-INST/DEL, e vi si rimane finche' non si sono premuti tanti tasti quanti erano gli spazi inseriti. Ovviamente si puo' uscire dal modo INSERT anche nei modi con cui si esce dal modo QUOTE (escluso chiudere le virgolette).

1.4 IL VIDEO

Il video e' il piu' immediato dispositivo di output, cioe' di uscita; esso puo' visualizzare 1000 caratteri: 25 righe di 40 caratteri ciascuna. Nel COMMODORE PLUS-4 esistono due insiemi di caratteri raggruppati nel SET MAIUSCOLO/GRAFICO e nel SET MINUSCOLO/MAIUSCOLO. Per passare da un SET all'altro devi premere contemporaneamente i tasti CBM LOGO e SHIFT. Esistono comunque 3 tipi di caratteri:

a) numerici
b) alfabetici
c) grafici.

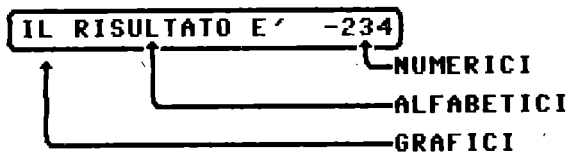



Figura 1.1 Tipi di carattere

I caratteri di tipo grafico possono essere usati per abbellire o rendere piu' leggibili le stampe dei risultati delle elaborazioni del tuo COMMODORE PLUS-4 o per cominciare a creare qualche semplice giochino o animazione (come vedrai piu' avanti).
Quando accendi il calcolatore, sul video vedi uno SCHERMO bianco, attorniato da un BORDO azzurro, delle scritte nere (del set MAIUSCOLO/GRAFICO) e, poco piu' sotto le scritte, un quadratino nero che lampeggia: il CURSORE. Il cursore indica il punto in cui verra' scritto il prossimo carattere e puo' essere spostato grazie ai quattro tasti di movimento del cursore in basso a destra sulla tastiera e a forma di freccia. Se sposti il cursore su delle scritte gia' esistenti, queste non verranno cancellate, ma il cursore segnalera' la sua presenza alternando, in quella posizione, il carattere stesso e il corrispondente carattere in campo inverso.

I COLORI E GLI EFFETTI SPECIALI

Il COMMODORE PLUS-4 e' un calcolatore che ha la possibilita' di visualizzare a colori: puoi scegliere i colori che preferisci per il bordo, lo sfondo e le scritte. Il modo piu' semplice per cambiare il colore del cursore, e quindi di tutti i caratteri che scriverai da quel momento in poi, e' quello di premere insieme i tasti CTRL e un tasto da 1 a 8 o CBM LOGO e un tasto da 1 a 8. Nel primo caso il cursore diventera' del colore scritto, in inglese, in alto, sulla parte frontale del tasto numerico premuto; nel secondo caso del colore scritto in basso. Premendo CTRL e "," tutti i caratteri, da quel momento in poi, vengono scritti lampeggianti (hai attivato FLASH ON); questo effetto dura fino a che non viene premuto RETURN o CTRL e "." (FLASH OFF).
Premendo CTRL e 9 (che attiva RVS ON), invece, i caratteri verranno scritti in campo inverso fino a che non premi RETURN o CTRL e 0 (RVS OFF).
Scegliendo il colore del cursore con i tasti CTRL e CBM LOGO puoi scegliere tra 16 colori, ma i colori del COMMODORE PLUS-4 sono di piu': ben 121!
Come fare, quindi, per scegliere tra tutti questi colori? Devi usare il comando COLOR del BASIC.
Se tu scrivi, su una linea dove non esistono altre scritte:
COLOR 1,7,6
e quindi premi RETURN, il calcolatore ti risponde "READY." in blu.

Il primo numero, che segue il comando COLOR, indica quale colore e' da cambiare:

0 cambia il colore dello schermo
1 cambia il colore del cursore
2 cambia il colore multicolore 2 (vedi Paragrafo 2.8)
3 cambia il colore multicolore 3 (vedi Paragrafo 2.8)
4 cambia il colore del bordo

nel nostro caso il primo numero e' 1, quindi e' stato cambiato il colore del cursore (e la scritta "READY." e' stata scritta col nuovo colore).
Il secondo numero indica il colore scelto:

1	nero	9	arancione
2	bianco	10	marrone
3	rosso	11	giallo-verde
4	ciano	12	rosa
5	porpora	13	verde-blu
6	verde	14	blu chiaro
7	blu	15	blu scuro
8	giallo	16	verde chiaro

nel nostro caso il secondo numero e' 7, quindi il colore scelto e' il blu.
L'ultimo numero indica la luminosita' del colore. Puoi scegliere un numero tra 0 (scurissimo) e 7 (chiarissimo); nel nostro esempio abbiamo scelto 6 cioe' molto chiaro.
Nota che il colore blu e il colore blu chiaro o il verde e il verde chiaro, anche se usati con la stessa luminosita', non sono identici. I colori disponibili sono quindi 15 (dal 2 al 16) X 8 (le luminosita') piu' il nero (per il quale non valgono le luminosita'); in tutto dunque 121!

1.5 IL COMMODORE PLUS-4 COME CALCOLATRICE

Puoi usare il tuo COMMODORE PLUS-4 per eseguire calcoli anche complicati.
Quando usi il calcolatore in questo modo, ti servi del linguaggio BASIC in modo IMMEDIATO, cioe' scrivi una istruzione e la esegui quando premi il tasto RETURN. Le istruzioni che puoi usare sono:
. PRINT, che puo' essere scritta anche come ?, cioe' il calcolatore interpreta il ? come se tu avessi scritto la parola PRINT; questa istruzione fa scrivere sul video quello che segue, se segue un'espressione, viene scritto il risultato del calcolo.
. ISTRUZIONE DI ASSEGNAZIONE, che si scrive:
  nome-variabile=espressione.
Essa assegna alla variabile il valore dell'espressione; se la variabile non esiste gia' viene creata. Il valore assegnato alla variabile rimane immutato fino a quando si esegue un'altra istruzione di assegnazione usando lo stesso nome, oppure si impartisce una istruzione che azzera tutte le variabili (come CLR).
Dopo aver assegnato un valore ad una variabile questa puo' essere utilizzata in qualunque espressione.

Puoi provare a scrivere:
PRINT 123*2-18*7
dopo la pressione del tasto RETURN vedi il risultato sulla linea seguente; oppure scrivi:
? 15*12-134*5
in questo caso il risultato e' preceduto dal segno meno, infatti esso e' negativo. Nel primo calcolo il risultato positivo e' preceduto da uno spazio.

Le ESPRESSIONI sono formate da costanti (numeri), da variabili, da funzioni, da parentesi rotonde e da operatori aritmetici. Esse devono essere scritte utilizzando le regole della matematica.

Gli OPERATORI ARITMETICI sono:

^ elevamento a potenza	¬ segno
* moltiplicazione	/ divisione
+ addizione	¬ sottrazione.

Gli elementi costanti, variabili e funzioni, devono essere colle¬gati tra loro mediante gli operatori aritmetici e le parentesi rotonde.
Non si possono avere due operatori aritmetici vicini, salvo nel caso del segno ¬. Si puo' scrivere: 6*¬8
ma questo risulta poco chiaro, per cui e' meglio scrivere: 6*(¬8).
Non si puo' scrivere 6*/8, perche' non ha senso usare i due operatori * e / vicini.
I numeri non interi si scrivono usando il punto decimale al posto della virgola.
Le VARIABILI sono dei CONTENITORI di dati; ad esse vengono asse¬gnati dei NOMI secondo regole precise.
Il nome di una variabile puo' essere formato da due caratteri, dei quali il primo deve essere una lettera e il secondo puo' essere una lettera o una cifra numerica e puo' anche mancare.
Prova a scrivere:

```
PRINT (18*(15¬3))^2
```

vedrai il risultato: 435673.001
Il calcolatore ha eseguito i calcoli cosi':

```
15-3=12             18*12=216
216^2=46656.0001
```

l'ultimo calcolo e' strano: il piccolo errore e' dovuto al fatto che il calcolatore, nell'eseguire alcune operazioni, come l'elevamento a potenza, deve eseguire molti calcoli intermedi approssimandone i risultati.

Prova ora a scrivere:

```
A=15*8
PRINT 1245¬A
```

ottieni come risultato 1125, infatti la variabile A contiene 120 (15*8) dato che tale valore le e' stato assegnato prima di usar¬la nell'espressione di calcolo.

Negli esempi di calcolo abbiamo parlato di numeri in generale; in realta' il COMMODORE PLUS¬4 puo' trattare due tipi di numeri:
. i NUMERI INTERI,
. i NUMERI REALI (quelli non interi).
Per distinguere il nome di una variabile che puo' contenere solo numeri interi da quello di una variabile che puo' contenere nume¬ri reali, si fa seguire al nome il suffisso %. La variabile A% contiene un numero intero, si chiama anche variabile intera. La

variabile A contiene un numero reale, si chiama anche variabile reale.
Se in una operazione di assegnazione a sinistra dell'uguale compare una variabile intera, il risultato dell'espressione viene reso intero prima di assegnarlo alla variabile. Per esempio:
B%=3.5, assegna a B% il valore intero 3.

I numeri interi purtroppo sono molto limitati in grandezza, cioe' una variabile intera puo' contenere un numero compreso tra -32768 e +32767. Se si cerca di assegnarle un numero fuori da questo intervallo, esso viene troncato perdendo il suo valore e ovviamente i risultati dei calcoli sono errati.
Nel calcolatore viene usata una rappresentazione approssimata per i numeri reali (cioe' con un numero limitato di cifre decimali dopo il punto decimale).
Nel COMMODORE PLUS-4 i numeri reali sono conservati con circa 10 cifre significative; il valore varia tra -4294967304 e +4294967304.
Essi possono essere rappresentati in uscita in due modi:
. in VIRGOLA FISSA,
. in VIRGOLA MOBILE (FLOATING-POINT) o FORMA ESPONENZIALE.
Il COMMODORE PLUS-4 decide di evidenziare un numero in uno dei due formati in base al numero delle cifre significative presenti. Fino a 9 cifre, ed escludendo un piccolo intervallo intorno allo zero (precisamente i numeri X che risultano: -0.01<X<0 oppure 0<X<0.01), il numero viene evidenziato in virgola fissa; gli altri numeri vengono evidenziati in virgola mobile. Nella Figura 1.2 sono indicati gli intervalli validi per le due rappresentazioni.

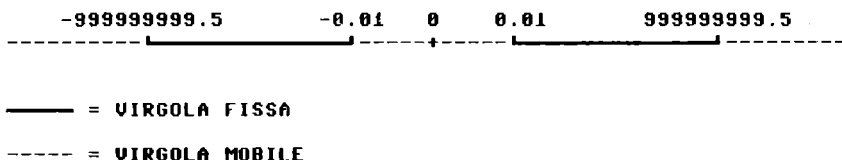



Figura 1.2 Intervalli rappresentazione numeri

Il ricorso alla rappresentazione in virgola mobile consente di limitare il numero di caratteri.
Infatti se possiamo scrivere un numero in una casella di un foglio di carta formata da 20 caselline, ognuna capace di registrare una cifra, non possiamo scrivere piu' di 20 cifre. In tale caso un numero come:
0.000000000000001234567891234567 8 non sappiamo come scriverlo;

infatti se scriviamo le prime 20 cifre abbiamo una buona approssimazione sulla grandezza del numero, ma perdiamo un bel gruppo di cifre significative, mentre se scriviamo le ultime 20 cifre cambiamo completamente la natura del numero, che diventa grande.
Analogamente un numero come:
12345678912345678900000000000, scritto in 20 cifre cambia comple¬ tamente ordine di grandezza.

Se decidiamo di usare le 20 caselline a disposizione in questo modo:
. le prime 3 per il segno e l'esponente da dare a 10 per rappre¬ sentare l'ordine di grandezza del numero (+xx o ¬xx); arriviamo a ordini di grandezza da 10 elevato a ¬99 (piccolissimo) a 10 ele¬ vato a +99 (molto grande),
. le altre 17 per il segno e le cifre significative del numero,
per ottenere il valore del numero moltiplichiamo le 17 cifre conservate (16 piu' il segno) per l'opportuna potenza di 10 (10^esponente). In questo caso conserviamo l'ordine di grandezza del numero e il maggior numero possibile di cifre significative. Inoltre e' inutile visualizzare gli zeri a sinistra o a destra in quanto ne puo' tener conto l'esponente.
Da quanto detto viene giustificata la terminologia "in virgola mobile"; infatti la virgola viene spostata dopo la prima cifra significativa del numero, come se tutti i numeri fossero del tipo: x.xx...x; viene tenuto conto degli spostamenti della vir¬ gola (che e' il punto decimale) nell'esponente.

Per i due numeri visti sopra:

0.0000000000000012345678912345678
prendendo solo 17 cifre significative:
+1.2345678912345678 * 10^¬15

12345678912345678900000000000
prendendo solo 17 cifre:
+1.2345678912345678 * 10^+27

Il COMMODORE PLUS¬4 fa proprio un lavoro di questo tipo per con¬ servare i numeri reali nella sua memoria, cioe' li conserva sem¬ pre in virgola mobile.
Il numero di cifre consentite e' diverso da quello citato nell'e¬ sempio, e, inoltre, il calcolatore lavora al suo interno con l'aritmetica binaria. Nell'aritmetica binaria sono usate solo due cifre, 0 e 1, e il valore posizionale delle cifre viene calco¬ lato in base alle potenze di 2. Tu non hai bisogno di occuparti dell'aritmetica binaria, infatti il COMMODORE PLUS¬4 comunica con l'esterno con la normale aritmetica decimale.

In uscita il COMMODORE PLUS-4 ci mostra i numeri reali in virgola fissa se hanno fino a 9 cifre (escludendo gli intervalli intorno allo zero sopra citati) e in virgola mobile negli altri casi. Per rendere evidente quest'ultimo tipo di rappresentazione il numero viene mostrato nella forma:

```
 X.XX...XE+YY     X.XX...XE-YY
-X.XX...XE+YY    -X.XX...XE-YY
```

e il valore di YY puo' essere al massimo 38, mentre le cifre X..-.X possono essere al massimo 9.

Le FUNZIONI corrispondono a un insieme di operazioni che vengono eseguite citando il nome dalla funzione seguito tra parentesi dall'ARGOMENTO da usare nel calcolo. L'argomento puo' essere una costante, una variabile o una espressione, purche' il valore sia tale da non contraddire la logica della funzione, tipo estrazione della radice quadrata di un numero negativo.

Le OPERAZIONI ARITMETICHE si ottengono usando gli operatori aritmetici. Il simbolo "=" ha il normale significato di uguaglianza usato in matematica, con qualcosa in piu'. Il normale significato di uguaglianza e' che quanto sta a sinistra del simbolo e' uguale a quanto sta a destra. In programmazione il simbolo "=" puo' essere usato in questo modo per indicare una relazione tra due entita', che puo' essere vera o non vera, oppure con il significato di assegnazione, cioe' alla variabile che sta a sinistra del simbolo viene assegnato il valore che risulta dall'espressione che sta a destra, espressione che puo' ridursi a una semplice costante. In questa ottica si puo' scrivere: X=X+1, che dal punto di vista matematico e' un non senso, ma dal punto di vista della programmazione significa "aggiungi 1 al precedente contenuto della variabile X".

Le ESPRESSIONI vengono calcolate partendo da sinistra e muovendosi verso destra, ma dando la precedenza ai calcoli indicati tra parentesi rotonde. Inoltre tra gli operatori aritmetici le precedenze sono:

. elevamenti a potenza,
. assegnazioni del segno meno,
. moltiplicazioni e divisioni,
. somme e sottrazioni.

Se nelle espressioni compaiono delle funzioni, esse sono calcolate subito e viene sostituito ad esse il loro valore. Se l'argomento delle funzioni e' un'espressione, essa viene calcolata prima della funzione.

CAPITOLO 2

PROGRAMMARE IN BASIC

2.1 IL PROGRAMMA

Un PROGRAMMA per calcolatore e' formato da una sequenza di istruzioni eseguibili da parte del calcolatore.
Ogni calcolatore e' costruito in modo da essere in grado di eseguire in modo automatico un programma scritto nel suo LINGUAGGIO MACCHINA e preventivamente registrato nella sua MEMORIA, secondo prescritte modalita'.
La redazione di programmi in linguaggio macchina e' abbastanza complessa e viene considerata un lavoro per specialisti; non si tratta comunque di un lavoro trascendentale, e, a nostro avviso, chiunque dotato di media intelligenza, pazienza e buona volonta' puo' imparare a programmare un calcolatore in linguaggio macchina.
A partire dall'avvento dei primi calcolatori, nella seconda meta' degli anni 50, e' stato affrontato il problema di mettere a punto linguaggi simbolici di programmazione, adatti a consentire la diffusione della pratica della programmazione e un conseguente utilizzo di massa dei calcolatori. Si trattava di inventare dei linguaggi adatti ad essere appresi dall'uomo, e basati su regole tali da consentirne senza ambiguita' la conversione in linguaggio macchina. Sono stati creati molti tipi diversi di linguaggi, ognuno con pregi e difetti, ma tutti molto utili alla diffusione dei calcolatori.

La memoria RAM (Random Access Memory, memoria ad accesso casuale, da interpretare come memoria per lettura e scrittura), del COMMODORE PLUS-4 serve per registrare i programmi e i dati dell'utente. La parte di programmi preregistrata (ROM, Read Only Memory) comprende il SISTEMA OPERATIVO e l'INTERPRETE BASIC. Il SISTEMA OPERATIVO comprende tutti i programmi che sovraintendono al funzionamento del calcolatore; esso e' formato da diversi moduli, ognuno con funzioni specifiche, come, per esempio, la gestione delle periferiche. Al momento dell'accensione del calcolatore e' attivo il modulo (EDITOR) che gestisce il colloquio con l'utente in modo strettamente connesso al lin-

guaggio BASIC. Il COMMODORE PLUS-4, come moltissimi altri calcolatori personal, e' predisposto alla programmazione in BASIC. E' possibile programmare il calcolatore anche in linguaggio macchina, ma l'accesso al linguaggio macchina si ha sempre partendo dal BASIC.
Dal momento che il calcolatore capisce solo il suo linguaggio macchina, per poter eseguire un programma scritto in un altro linguaggio e' necessario che sia presente nella memoria del calcolatore un programma, scritto in linguaggio macchina, che funga da intermediario e traduca le istruzioni BASIC in istruzioni eseguibili. Tale intermediario e' l'INTERPRETE BASIC.
La scritta che compare all'accensione del calcolatore dice che il COMMODORE PLUS-4 contiene l'interprete BASIC nella versione 3.5, e che tu puoi disporre di 60671 byte di memoria RAM per il tuo programma.

Indipendentemente dal linguaggio di programmazione usato e' necessario capire cosa e' un programma per calcolatore. Esso e' una sequenza di istruzioni elementari che descrivono, passo dopo passo, cosa il calcolatore deve fare, in modo che:

. partendo da dati iniziali,
. svolga un certo tipo di trasformazione dei dati,
. produca dei risultati.

In sostanza un programma opera una trasformazione di dati, intendendo questo concetto in modo molto generale. Possiamo considerare una trasformazione di dati cose come:

. risolvere un'equazione di secondo grado,
. calcolare la lunghezza della circonferenza di un cerchio,
. calcolare un volume,
. elaborare e stampare una fattura,
. mettere in ordine alfabetico dei nominativi,
. ricercare quante volte un determinato aggettivo compare in un testo,
. giocare a scacchi,

e quante altre se ne vogliano aggiungere.
Qualunque programma tu voglia scrivere, la prima cosa da fare e' conoscere bene il problema che il programma deve affrontare e risolvere; questo e' di gran lunga il compito piu' difficile per il programmatore.

Un programma va organizzato procedendo nello studio del problema che si vuole risolvere.
Il primo passo e' la definizione esatta e completa del problema che deve essere affrontato. In generale, dopo una prima definizione e l'inizio della relativa riflessione, il problema puo' anche essere ridefinito in modo piu' opportuno.
A questo punto e' necessario organizzare in modo preciso i dati sui quali il programma deve lavorare. Dei dati interessano diver-

si aspetti:
. la loro natura,
. la loro quantita',
. la loro fonte.
Si deve arrivare alla stesura di uno schema contenente tutte le caratteristiche dei dati di ingresso, chiamati dati di INPUT.
Esaurito l'argomento dei dati di INPUT, deve essere affrontato quello dei risultati, cioe' dei dati di OUTPUT:
. cosa si vuole ottenere,
. in quale forma,
. su quale mezzo.
Nel caso di una stampa su carta, deve, per esempio, essere preci-sato cosa scrivere su ogni riga, a quale distanza deve trovarsi ogni dato dal precedente, e cosi' via.
Avendo chiarito da dove si parte e dove si vuole arrivare, e' necessario stabilire come arrivarci. Si tratta cioe' di sceglie-re la procedura da seguire; si dice anche scegliere gli algo-ritmi piu' adatti. Con la parola ALGORITMO si intende "modo di procedere", cioe' la sequenza di operazioni necessarie per ope-rare la desiderata trasformazione di dati.
Abbiamo parlato di operazioni; esse possono essere di tipo matematico o logico, e con esse si puo' costruire qualunque programma, anche molto complesso.

Lo studio del problema comporta la stesura di note, che possono essere organizzate in modi diversi. Alcuni preferiscono degli appunti schematici, nei quali i successivi passi da compiere ricevono una numerazione progressiva. Altri preferiscono ricor-rere alla stesura di diagrammi, cioe' di schemi grafici, formati da blocchetti di diversa forma, ognuno con un particolare significato, all'interno dei quali sono scritte note sintetiche esplicative.
In generale si consiglia un metodo TOP-DOWN, nel quale, partendo da una definizione molto generale del problema, si scende, per affinamenti successivi, alla stesura finale, che assomiglia gia' molto alla CODIFICA (scrittura) del programma nel linguaggio di programmazione scelto.

Tutto il lavoro che deve essere svolto prima di arrivare alla codifica di un programma va sotto il nome di: ANALISI DEL PROBLEMA.
Ti potrai chiedere se quando inizi lo studio di un problema devi decidere a priori in quale linguaggio verra' codificato il programma. La risposta e' che questo non e' a priori necessario, anche se la scelta del linguaggio influisce sulla semplicita' della programmazione e sull'efficienza dell'elaborazione. E' comunque certo che, se un problema e' stato ben studiato, esso puo' essere tradotto facilmente in un programma scritto in un qualunque linguaggio.

I dati che vengono trattati da un programma possono essere costanti o variabili. I dati costanti possono essere introdotti direttamente nelle linee di programma come numeri interi o decimali, scritti nel formato a virgola fissa o a virgola mobile, o come sequenze di caratteri alfanumerici, che vengono chiamate STRINGHE. I dati, che variano durante l'esecuzione del programma, come quelli che si ricevono in INPUT, i risultati intermedi o finali, costituiscono le VARIABILI del programma. Ogni VARIABILE viene definita assegnandole un nome simbolico, che rispetti le regole del linguaggio. Il dato viene richiamato citando il nome della variabile che lo contiene.
Quando accendi il COMMODORE PLUS-4 esso e' gia' pronto e aspetta che tu scriva un programma o impartisca istruzioni da eseguire in modo immediato.
E' attivo l'EDITOR, che ti consente l'uso del video e della tastiera (vedi Paragrafo 1.3).
Se vuoi essere sicuro che la zona di memoria dedicata al programma sia pulita, devi scrivere:
NEW e premere RETURN.
Per effetto di questo comando, impartito qui in modo immediato, la memoria viene pulita dal programma e dai dati eventualmente presenti.
Se desideri vedere un video pulito e cominciare a scrivere in alto, puoi premere i tasti SHIFT+CLEAR/HOME.
A questo punto puoi scrivere le tue linee di istruzioni numerandole progressivamente. Se ti sbagli e dimentichi una linea, e' semplice rimediare, basta scrivere la linea dimenticata iniziando con un numero tale che le consenta di essere posizionata dove desideri.
Mentre procedi nella scrittura delle linee, il video si riempie e puo' aver luogo lo scorrimento delle scritte verso l'alto (scrolling).
Se ti accorgi che qualche linea presente sullo schermo e' errata puoi riscriverla ex novo o spostare il cursore, per mezzo dei relativi tasti, posizionarti dove desideri e correggere; la linea modificata, o riscritta, va a sostituirsi alla precedente al momento della pressione del tasto RETURN.
Se desideri abolire una linea basta scriverne il numero e premere RETURN.
Se desideri rivedere una parte di programma che e' scomparsa dal video per effetto dello scrolling, puoi chiederne la lista con il comando LIST. Solo che se scrivi LIST e premi RETURN e il programma che sta in memoria ha piu' di 25 linee perdi le prime. Puoi usare una forma diversa del comando LIST; scrivere per esempio: LIST -100, e ottieni la lista delle linee di programma fino a quella numerata 100. Se invece scrivi: LIST 40-120, ottieni la lista delle linee numerate da 40 a 120. Se scrivi: LIST 80, ottieni la lista della sola linea 80. Se scrivi: LIST 110-,

ottieni la lista delle linee dalla 110 in avanti.
Il tuo programma rimane in memoria fino a quando non scrivi NEW oppure togli corrente al calcolatore.

2.2 IL LINGUAGGIO BASIC

Il linguaggio BASIC e' di tipo:
. SIMBOLICO,
. INTERPRETATIVO,
. CONVERSAZIONALE.

SIMBOLICO, perche' lavora su simboli facilmente comprensibili, che possono essere parole chiave del linguaggio e nomi simbolici creati dal programmatore.
INTERPRETATIVO, perche' durante l'esecuzione del programma e' presente nella memoria del calcolatore il programma INTERPRETE BASIC, che provvede alla interpretazione, traduzione e esecuzione delle frasi del linguaggio.
CONVERSAZIONALE, perche' si svolge un colloquio tra calcolatore e utente, che rende molto semplice la stesura, la correzione e l'esecuzione del programma.

Possiamo schematizzare questo grande e potente foglio elettronico, che e' la memoria del nostro COMMODORE PLUS-4, immaginandola divisa nelle seguenti parti:
. ROM SISTEMA OPERATIVO,
. ROM INTERPRETE BASIC,
. RAM programma utente in BASIC,
. RAM dati del programma, del SISTEMA OPERATIVO e dell'INTERPRETE BASIC,
come indicato in Figura 2.1.

SISTEMA OPERATIVO
INTERPRETE BASIC
PROGRAMMA UTENTE IN BASIC
DATI DEL PROGRAMMA UTENTE

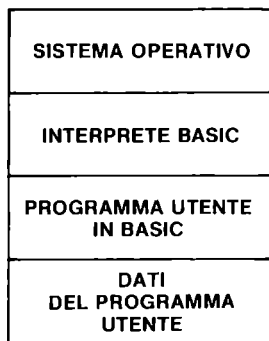



Figura 2.1 Schema della memoria

Esistono molte versioni del linguaggio BASIC; e' comune a tutte la stessa filosofia di gestione delle risorse del calcolatore, ma possono esserci anche molte differenze, tali da rendere una versione molto piu' potente di un'altra.
Il COMMODORE PLUS-4 e' dotato della versione 3.5 del BASIC, derivata dalla versione originale della MICROSOFT, con aggiunta di nuovi comandi, che la rendono molto potente. E' molto interessante avere su un personal di basso costo una cosi' ricca versione del BASIC, come questa marcata 3.5.

Le istruzioni del BASIC sono formate da PAROLE CHIAVE del linguaggio e da PARAMETRI. Con PAROLE CHIAVE si intendono una o piu' parole che hanno un particolare significato operativo. Con PARAMETRI si intende tutto quello che deve essere scritto oltre le parole chiave per rendere completa un'istruzione.
Le istruzioni del linguaggio sono organizzate in linee che possono comprendere piu' di una istruzione. Se una linea comprende piu' istruzioni, esse devono essere scritte separandole con il carattere separatore "due punti" (:). Il carattere ":" ha per il BASIC il significato di separatore.
Una linea di istruzioni puo' essere scritta ponendo inizialmente un numero: NUMERO DI LINEA, oppure iniziando subito con la prima istruzione. Questo diverso modo di scrivere una linea da' luogo a un comportamento diverso al momento della pressione del tasto RETURN, che chiude la linea. Se la linea inizia con il NUMERO DI LINEA, essa viene memorizzata nella zona di memoria dedicata al programma utente, assegnandole la giusta posizione in base al numero di linea. Le linee di istruzioni vengono memorizzate per numero di linea crescente. Se la linea non inizia con un numero essa viene eseguita immediatamente al momento della pressione del tasto RETURN.
Il BASIC puo' dunque lavorare in due modi:
. MODO IMMEDIATO (linee senza numero),
. MODO DIFFERITO (linee numerate).
Vedremo procedendo che non tutte le istruzioni possono essere eseguite nei due modi.
Abbiamo gia' fatto esperienza del modo di esecuzione immediato nel paragrafo 1.5.
Nell'Appendice A e' riportata la scheda del BASIC 3.5 del COMMODORE PLUS-4 e ad essa ti rimandiamo per una visione completa del linguaggio.
Raggruppiamo le istruzioni in classi, per avere uno schema sintetico delle possibilita' del linguaggio.
Abbiamo:
..Istruzioni per l'ingresso dei dati (operazioni di INPUT): DATA, GET, GET#, GETKEY, INPUT, INPUT#, JOY, READ, RESTORE.
..Istruzioni per l'uscita dei dati (operazioni di OUTPUT): CHAR, POS, PRINT, PRINT#, USING, PUDEF, SPC, TAB.

..Istruzioni e funzioni di calcolo: ABS, ATN, COS, DEC, DEF FN, EXP, FN, INT, LET, LOG, RND, SGN, SIN, SQR, TAN, VAL.
..Istruzioni e funzioni per dati alfanumerici: ASC, CHR$, HEX$, INSTR, LEFT$, LEN, MID$, RIGHT$, STR$.
..Istruzioni di controllo: DO...LOOP (UNTIL, WHILE, EXIT), END, GOTO, GOSUB, IF...THEN...ELSE, FOR...TO...STEP, NEXT, ON...GOTO, ON...GOSUB, RESUME, RETURN, SYS, TRAP, USR, WAIT.
..Istruzioni per grafica ad alta risoluzione e multicolore: BOX, CIRCLE, COLOR, DRAW, GRAPHIC, GSHAPE, LOCATE, PAINT, RCLR, RDOT, RGR, RLUM, SCALE, SCNCLR, SSHAPE.
..Istruzione per suonare: SOUND, VOL.
..Istruzioni di servizio per la stesura e la gestione del programma e dei dati: AUTO, BACKUP, CLOSE, CLR, CMD, COLLECT, CONT, COPY, DELETE, DIM, DIRECTORY, DLOAD, DSAVE, FRE, HEADER, KEY, LIST, LOAD, MONITOR, NEW, OPEN, PEEK, POKE, REM, RENAME, RENUMBER, RUN, SAVE, SCRATCH, STOP, TROFF, TRON, VERIFY.

Le regole per la formazione dei nomi delle variabili sono le seguenti:
. Il nome deve essere formato da al massimo due caratteri (ne puoi usare di piu', ma vengono riconosciuti solo i primi due), il primo deve essere una lettera, il secondo puo' essere una lettera o una cifra.
. Il nome deve essere seguito dal suffisso % per indicare numeri interi.
. Il nome deve essere seguito dal suffisso $ per indicare variabili stringa.
. I nomi senza suffisso si riferiscono a variabili reali. Le variabili sopra descritte sono VARIABILI SINGOLE, ogni nome corrisponde a un dato.

2.3 LE ISTRUZIONI PIU' ELEMENTARI

Ci proponiamo di risolvere il seguente problema:
CHIEDERE UN NUMERO, CALCOLARNE IL QUADRATO E IL CUBO E MOSTRARE I RISULTATI.

Per avere una certa liberta' dobbiamo lavorare con numeri reali, infatti la grandezza consentita per i numeri interi ci bloccherebbe subito. Dobbiamo scegliere un nome per la variabile che deve ricevere da tastiera il numero; la chiamiamo N. Il numero N e' l'unico dato di INPUT necessario per risolvere il nostro problema. Dobbiamo creare due nomi di variabili per contenere rispettivamente il quadrato e il cubo di N; scegliamo N2 per il quadrato e N3 per il cubo. L'algoritmo di calcolo e' molto semplice, infatti:
N2=N*N e N3=N2*N. Abbiamo gia' visto che l'asterisco si usa come operatore per la moltiplicazione. Le formule di calcolo posso-

no anche essere scritte in un altro modo:
N2=N^2 e N3=N^3, dove la freccia verso l'alto significa "elevato a".
Decidiamo di fornire i risultati in modo chiaro, scrivendo sul video:
N=...
IL QUADRATO DI N E' UGUALE A ...
IL CUBO DI N E' UGUALE A ...
Descriviamo verbalmente la procedura da programmare:
.1) legge dalla tastiera un numero N;
.2) calcola il quadrato di N e lo pone in N2;
.3) calcola il cubo di N e lo pone in N3;
.4) stampa i risultati.

Possiamo descrivere la procedura anche con lo schema riportato nella Figura 2.2, che prende il nome di DIAGRAMMA A BLOCCHI.

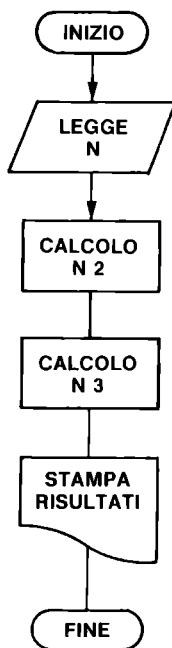



Figura 2.2 Diagramma a blocchi ES2.1

Riportiamo ora il listato del programma BASIC ES2.1 che risolve il problema posto.

```
1 REM ES2.1
10 INPUT"SCRIVI UN NUMERO ";N
20 N2=N↑2:N3=N↑3
30 PRINT"N = ";N
40 PRINT"IL QUADRATO DI N E' UGUALE A: ";N2
50 PRINT"IL CUBO DI N E' UGUALE A: ";N3
```

Ti facciamo osservare i limiti di questo programma; esso lavora per un solo numero, dopo aver eseguito i calcoli e stampato i risultati si ferma.
Commentiamo ora le linee del programma ES2.1.
. 1: inizia con la parola chiave REM, abbreviazione di REMark, che in italiano significa ANNOTAZIONI. Quello che segue REM viene considerato un commento; noi abbiamo scritto il nome attribuito al programma, ma potevamo scrivere qualunque altra cosa. I commenti possono contenere anche il carattere ":", che in questo caso non viene considerato come carattere separatore tra istruzioni; in conseguenza l'istruzione REM deve essere l'ultima di una linea che contenga piu' istruzioni.
. 10: e' l'istruzione per chiedere dati dalla tastiera; alla parola chiave INPUT puo' seguire, come nel nostro caso, un messaggio esplicativo della richiesta di dati, compreso tra virgolette, seguito da ";" e dalla lista delle variabili nelle quali si vuole siano memorizzati i dati letti dalla tastiera, separate da virgola. Il messaggio deve essere dato sotto forma di costante; non si puo' usare il nome di una variabile che lo contenga.
Il calcolatore evidenzia un "?" di richiesta dati (dopo il messaggio, se presente). Devi rispondere con tanti dati quanti sono quelli richiesti, ponendo una virgola alla fine del dato, se ne seguono altri, e premendo RETURN alla fine. Se rispondi con meno dati di quelli richiesti, il calcolatore va a capo e ti mostra "??" a segnalarti che non hai esaurito la richiesta. Devi cercare di rispondere con dati che concordino con il tipo delle variabili presenti nella lista; in caso contrario il calcolatore scrive "?REDO FROM START" e ti chiede nuovamente i dati. Hai questa segnalazione di errore se rispondi con una parola alla richiesta del numero nel nostro programma.
. 20: viene assegnato alla variabile N2 il quadrato del numero N e alla variabile N3 il cubo del numero N. La linea comprende due istruzioni di assegnazione separate da ":". L'istruzione di assegnazione puo' iniziare con la parola chiave LET, cosi': LET N2=N^2, che pero' e' facoltativa. Dopo L'esecuzione dell'istruzione, la variabile posta a sinistra di "=" contiene il risultato dell'espressione posta a destra.
. 30: la parola chiave PRINT ordina di stampare quello che segue. La lista di stampa puo' comprendere variabili e costanti, che possono essere separate da:

";", per stampare i dati nel loro formato senza aggiunta di spa-zi intermedi;
",", per stampare il dato passando alla prossima zona di stampa per il successivo. Sul video le zone di stampa sono 4 di 10 caratteri ciascuna.
Noi stampiamo un messaggio e il numero N.
Se la lista di stampa termina senza punteggiatura (, o ;), come nel nostro caso, si ha il passaggio a nuova linea, dopo la stam-pa (va a capo).
. 40: stampa un messaggio e il numero N2.
. 50: stampa un messaggio e il numero N3. I due risultati appari-ranno nel formato necessario per il numero di cifre che li compongono.

FORMATI DI STAMPA
I formati di stampa sono:
. uno spazio o il segno meno, il numero, il salto di una posizio-ne, per i dati numerici;
. i caratteri che le compongono per le stringhe.

Riportiamo ora il listato del programma ES2.2, che rappresenta una modifica di ES2.1. In esso abbiamo introdotto dei caratteri di controllo nei messaggi, e precisamente:
. 10: il carattere FLASH ON all'inizio del messaggio e il carat-tere FLASH OFF alla fine.
. 30: il carattere RVS ON all'inizio del messaggio e il carat-tere RVS OFF alla fine.
. 40: il carattere CTRL¬3, per passare al colore rosso e il carattere RVS ON all'inizio del messaggio, il carattere RVS OFF alla fine. In questo modo e' rimasto attivo il colore rosso anche per la stampa del numero.
. 50: i caratteri RVS ON e RVS OFF all'inizio e alla fine del messaggio.
. 60: abbiamo aggiunto PRINT del carattere CTRL¬1 per tornare al colore nero.

```
1 REM ES2.2
10 INPUT"▌SCRIVI UN NUMERO▄ ";N
20 N2=N↑2:N3=N↑3
30 PRINT"▣N= ▀";N
40 PRINT"▣▣IL QUADRATO DI N E' UGUALE A:▀ ";N2
50 PRINT"▣IL CUBO DI N E' UGUALE A:▀ ";N3
60 PRINT"▪"
```

Segue il listato del programma ES2.3, nel quale, rispetto ad ES2.1, abbiamo solo aggiunto la linea 60. Essa contiene una PRINT senza lista per andare a capo e l'istruzione GOTO10. L'istru-zione GOTO (che puo' anche essere scritta GO TO), seguita dal

numero di una linea del programma, provoca un salto incondizionato al numero di linea citato, cioe' fa continuare l'esecuzione del programma da quel punto.

```
1 REM ES2.3
10 INPUT"SCRIVI UN NUMERO ";N
20 N2=N↑2:N3=N↑3
30 PRINT"N = ";N
40 PRINT"IL QUADRATO DI N E' UGUALE A: ";N2
50 PRINT"IL CUBO DI N E' UGUALE A: ";N3
60 PRINT:GOTO10
```

Il programma ES2.3 e' un programma che non finisce mai. E' un po' laborioso interromperlo, infatti durante la richiesta di dati il calcolatore non riconosce lo STOP; puoi premere contemporaneamente RUN/STOP e il tasto laterale di RESET, per interrompere il programma senza cancellarlo dalla memoria, e poi X e RETURN.

Segue il programma ES2.4, che consiste nella stampa di costanti, numeri e stringhe, per farti comprendere bene come viene gestita la stampa sul video a seconda dei separatori usati nella lista di stampa.

```
1 REM ES2.4
10 S$="1234567890123456789012345678901234567890"
15 PRINTS$
20 PRINT"ABC","DEF","GHI"
25 PRINTS$
30 PRINT"ABC";"DEF";"GHI"
35 PRINTS$
40 PRINT35,-987,1237
45 PRINTS$
50 PRINT"AABBCCDDEEFFGGHHLLM","NUOVA"
55 PRINTS$
60 PRINT"AABBCCDDEEFFGGHHLLMM","NUOVA"
65 PRINTS$
70 PRINT2345;9876;-5432
```

In ES2.4 la stringa S$, che viene stampata prima di ogni lista di dati, serve per numerare le colonne del video; risulta cosi' piu' facile controllare l'effetto della lista di stampa successiva. Riportiamo i risultati di ES2.4.

```
1234567890123456789012345678901234567890
ABC       DEF       GHI
1234567890123456789012345678901234567890
ABCDEFGHI
1234567890123456789012345678901234567890
```

```
 35        -987       1237
1234567890123456789012345678901234567890
AABBCCDDEEFFGGHHLLM NUOVA
1234567890123456789012345678901234567890
AABBCCDDEEFFGGHHLLMM          NUOVA
1234567890123456789012345678901234567890
 2345  9876 -5432
```

Come puoi osservare sono verificate tutte le cose dette fino ad ora, e cioe' l'effetto della virgola e del punto e virgola.

Nel programma ES2.5, richiediamo 3 dati con una istruzione di INPUT. Puoi provare a rispondere premendo RETURN dopo ogni dato per vedere comparire "??".

```
1 REM ES2.5
10 INPUT"SCRIVI TRE NUMERI ";X1,X2,X3
20 N=(ABS(X1*X2-X3))↑(1/2)
25 M=SQR(ABS(X1*X2-X3)):Z=N-M
30 PRINT"X1="X1"X2="X2"X3="X3
40 PRINT"ABS(X1*X2-X3)↑(1/2)="N
50 PRINT"SQR(ABS(X1*X2-X3))="M
60 PRINT"N-M="Z
```

Riportiamo il commento a ES2.5.
. 10: richiesta dei 3 numeri X1,X2 e X3.
. 20: calcolo di N. Nell'espressione abbiamo usato ABS per evitare di elevare a esponente 1/2 (estrazione di radice) un numero negativo.
. 25: calcolo di M. Nell'espressione usiamo la funzione SQR per estrarre la radice, cioe' scriviamo in altro modo lo stesso calcolo di prima. Inoltre calcoliamo in Z la differenza tra N e M.
. 30: stampa i 3 dati di INPUT, preceduti ognuno da un messaggio, senza usare separatori nella lista di stampa.
. 40,50,60: stampa i risultati senza usare separatori nella lista di stampa.

Ricorda che se nelle istruzioni INPUT rispondi solo con RETURN alla richiesta di un dato, la variabile interessata mantiene il suo precedente valore. Inoltre, se prima di scrivere il dato premi la barra di spazio, questi spazi non vengono conservati. Per conservare spazi o altri caratteri, tipo i separatori, all'interno delle variabili stringa, devi iniziare aprendo le virgolette, e terminare chiudendole.

2.4 ESECUZIONE DEL PROGRAMMA

Per eseguire un programma basta scrivere RUN e premere RETURN. L'effetto del comando RUN e' il seguente:
. 1: vengono azzerate tutte le variabili numeriche,
. 2: vengono ridotte a stringhe nulle, di 0 caratteri, tutte le variabili stringa,
. 3: il programma va in esecuzione a partire dalla sua prima istruzione.

Il comando RUN puo' essere scritto anche in altro modo: RUN numero-linea. In questo caso i punti 1 e 2 sono identici, mentre il programma va in esecuzione a partire da numero-linea.

Se vuoi evitare l'effetto dei punti 1 e 2 puoi far partire il programma scrivendo:
GOTO num-linea e premendo RETURN.

Se durante l'esecuzione del programma vedi comparire un messaggio di errore, devi rivedere il listato, e in particolare la linea segnalata come errata, correggere e riprovare. Ricordati che modificando il programma si cancellano i contenuti delle variabili. Vedi il Paragrafo 3.8.

La prova di un programma e' una parte molto importante del lavoro di programmazione. Essa puo' risultare molto gravosa, se non e' stata svolta bene l'analisi del problema. Inoltre, se un programma e' complesso, devi dedicare molta cura alla preparazione dei casi prova; infatti essi devono essere tali da consentire di provare il programma in tutte le sue parti e nelle condizioni limite.
Per esempio, se nel programma ES2.5, avessimo omesso l'uso della funzione ABS e eseguito la prova solo con numeri tali che il risultato del calcolo fosse stato positivo, il programma avrebbe in seguito dato errore, la prima volta che i numeri scelti avessero fornito un risultato negativo.

2.5 SCRITTURA FACILITATA DEL PROGRAMMA

Il BASIC 3.5 mette a disposizione alcuni comandi che facilitano la stesura dei programmi. Essi sono: AUTO e RENUMBER.

Il comando AUTO, che ha senso usare solo in modo immediato, consente di fissare l'incremento per i numeri di linea che si scrivono dopo la sua esecuzione. Scrivendo:

```
AUTO 10
```

fissi a 10 l'incremento tra i numeri di linea. Dopo l'esecuzione

del comando, puoi scrivere una linea di programma iniziando con il numero che desideri; quando premi RETURN, la tua linea viene memorizzata e compare automaticamente sul video il numero della linea successiva (vecchio numero + 10, in questo caso), con il cursore posizionato dopo di esso, e tu puoi cominciare a scrivere la prima istruzione della nuova linea. In sostanza risparmi la fatica di scrivere il numero di linea e eviti di sbagliarlo. Conviene dare come incremento un numero maggiore di 1, in modo di poter inserire nuove linee in caso di bisogno.
Per uscire dalla predisposizione AUTO, basta scrivere AUTO e premere RETURN. Per interrompere la numerazione e passare ad altro basta premere solo RETURN quando compare il numero della nuova linea; questo pero' non annulla la predisposizione AUTO che rimane attiva e si manifesta quando si ricomincia a scrivere nuovamente una linea di programma o se ne corregge una gia' esistente.

Non devi preoccuparti se dopo aver immesso un programma esso si presenta con numerazione irregolare, cioe' non sempre lo stesso intervallo tra le linee; il programma infatti funziona ugualmente, se non contiene istruzioni errate o errori di impostazione. Quando sei sicuro che il programma funziona, oppure quando lo desideri, puoi rimettere ordine nella numerazione delle linee di programma con il comando RENUMBER.

Il comando RENUMBER, che ha senso usare solo in modo immediato, si puo' scrivere senza far seguire la parola chiave da parametri; in questo caso il programma presente in memoria, viene rinumerato partendo dal numero 10 per la prima linea e incrementando di 10 i numeri di linea. Nella rinumerazione vengono risistemati tutti i richiami a linee preesistenti. La parola chiave RENUMBER puo' essere seguita da 3 parametri:
RENUMBER numeron,inc,numerov
dove:
. numeron, e' il nuovo numero da cui partire,
. inc, e' l'incremento da usare tra i numeri di linea,
. numerov, e' il vecchio numero di linea da cui partire nella rinumerazione.
Se trascuri il primo o il secondo parametro, devi mettere al loro posto una virgola, e vengono presi i valori di default, cioe' 10. La presenza del terzo parametro consente di rinumerare il programma in modo parziale, cioe' partendo da un punto determinato e non modificando quello che viene prima. Non e' consentito usare per il primo parametro un valore uguale a un numero di linea gia' esistente, e che non viene modificato per effetto del punto di partenza (terzo parametro).

Durante la stesura di un programma RENUMBER puo' essere usato

anche piu' volte. Per esempio a un certo punto vuoi aggiungere tra due istruzioni preesistenti piu' istruzioni di quanto possibile in base alla numerazione; allora usi RENUMBER dando un incremento grande, aggiungi tutto quello che credi e poi usi ancora RENUMBER per fare ordine.

Quando un programma e' terminato e si pensa di non doverlo piu' modificare, e' consigliabile rinumerarlo partendo da 1 e usando un incremento di 1; evitando i numeri grandi si risparmia memoria.
Nel Paragrafo 2.1 abbiamo gia' parlato dei comandi NEW e LIST. Ti suggeriamo ora alcuni accorgimenti che ti aiutano nella stesura di un programma. Se il tuo programma contiene istruzioni simili o uguali, puoi richiamare sul video con LIST n, la linea di modello, poi portarti su di essa e modificarla dove necessario, anche solo nel numero di linea. Sei facilitato dall'uso dei tasti di spostamento del cursore e dalle possibilita' che ti offre l'EDITOR (vedi Paragrafo 1.3).

2.6 GESTIONE DEL PROGRAMMMA

E' importante conservare i programmi su cassetta per poterli riutilizzare quando servono. L'unita' DATASSETTE 1531 consente di posizionare il nastro al numero di giri desiderato e questo e' utile per un buon utilizzo dello stesso. Ti raccomandiamo di mantenere i tasti del registratore in posizione di riposo e di azionarli solo quando richiesto.

Per memorizzare su nastro un programma devi scrivere:
SAVE "nome"
dove "nome" e' il nome che vuoi assegnare al programma per riconoscerlo. Il nome del programma puo' anche essere dato sotto forma di variabile stringa.
Dopo aver premuto RETURN, sul video compare il messaggio:
PRESS PLAY & RECORD ON TAPE
devi eseguire; basta premere RECORD, dato che si abbassa insieme anche PLAY.
Il calcolatore risponde con:
OK
SAVING "nome"
e il video si sbianca diventando del colore del bordo, il nastro gira, si accende l'indicatore luminoso del registratore, dopo poco ricompaiono le scritte precedenti sul video, con in piu':
READY.
si spegne l'indicatore del registratore e il nastro si ferma. A questo punto il tuo programma e' stato registrato ed e' consigliabile rimettere i tasti PLAY e RECORD in posizione di riposo.

Per essere sicuro che la registrazione e' stata buona, devi procedere alla verifica operando cosi':
. riavvolgere il nastro per posizionarlo all'inizio della registrazione,
. scrivere:
 VERIFY "nome" e premere RETURN,
il calcolatore chiede:
 PRESS PLAY ON TAPE
e dopo la pressione del tasto compare:
 OK
 SEARCHING FOR "nome"
si sbianca il video , dopo un po' compare:
 FOUND "nome"
 VERIFYING
si sbianca nuovamente il video e alla fine compare:
 OK, se e' andato tutto bene,
 ?VERIFY ERROR se la verifica non e' stata buona.
In quest'ultimo caso si deve riavvolgere il nastro al punto giusto e ripetere la procedura di memorizzazione con conseguente verifica.

Se ometti il nome dopo VERIFY viene confrontato il primo programma trovato sul nastro con quello presente in memoria. Usando il nome, invece, se vengono incontrati prima altri programmi, essi non vengono confrontati e la verifica avviene solo quando viene trovato il programma con il nome richiesto.
Segue la spiegazione del significato del FLAG che puo' accompagnare le istruzioni SAVE, VERIFY e LOAD. L'argomento e' stato affrontato per completezza, pero', se sei un principiante, ti consigliamo di tralasciarne per il momento la lettura.
Abbiamo considerato la forma piu' semplice del comando SAVE; in realta', dopo il nome del programma, si possono aggiungere altri due parametri, cosi':
SAVE "nome",1,flag
dove 1 e' il numero logico dell'unita' DATASSETTE 1531, e il flag puo' valere 1, 2 o 3, con il seguente significato:
. flag=1, per memorizzare il programma in modo che al momento del LOAD non venga modificato il suo indirizzo di inizio in memoria, ma mantenga quello che era al momento del SAVE,
. flag=2, per memorizzare dopo il programma una segnalazione di "FINE NASTRO",
. flag=3, per ottenere insieme i due precedenti effetti.
Se il flag viene omesso esso vale 0 e ha il significato di memorizzare il programma in modo rilocabile. La RILOCAZIONE di un programma ha questo significato:
. di norma il programma BASIC inizia al byte di indirizzo 4097 e questo indirizzo si trova nei byte 43 e 44 (puntatore all'inizio del programma),
. se prima di scrivere un programma sposti il puntatore all'i-

nizio del programma modificando il contenuto dei byte 43 e 44, il programma inizia a un indirizzo diverso da 4097,
. se usi il flag=0 nell'istruzione SAVE il programma viene memorizzato in modo da essere rilocabile,
. se usi il flag=1 o il flag=3 nell'istruzione SAVE il programma viene memorizzato mantenendo l'informazione sul suo indirizzo di inizio, qualunque esso sia, e non e' rilocabile,
. al momento del LOAD, istruzione nella quale e' possibile usare per il flag il valore 1, o il valore 0 (assenza di flag), si ha il seguente comportamento:
.. per flag=0 il programma viene caricato a partire dall'indirizzo di inizio programma che si trova nei byte 43 e 44, rilocandolo o meno a seconda del modo come era stato memorizzato
.. per flag=1 il programma viene caricato a partire dall'indirizzo dove si trovava al momento del SAVE, e questo puo' essere in disaccordo con il valore contenuto nei byte 43 e 44.

Se per memorizzare e' stato usato il flag, l'istruzione VERIFY deve essere scritta in modo tale che non ci sia contrasto con la situazione della memoria e quello che sta sul nastro. Se risulta necessario usare il flag, puoi scrivere cosi':
VERIFY "nome",1,flag.
Se sottointendi il numero dell'unita', viene assunto il valore di default che e' 1. Se vuoi scrivere il flag e non mettere 1, devi usare due virgole.
Per caricare in memoria un programma si usa il comando LOAD; nella forma completa esso si scrive:
LOAD "nome",1,flag
dove:
. "nome", puo' anche essere una variabile stringa che contenga il nome del programma,
. 1, e' il numero logico dell'unita' DATASSETTE 1531,
. flag puo' valere 0 (o essere assente) o 1, con i significati sopra spiegati.

2.7 STRUTTURE DEL PROGRAMMA

Fino ad ora abbiamo esaminato programmi con svolgimento sequenziale. Ci occupiano qui delle strutture di controllo, che permettono di uscire dalla sequenza e proseguire da altri punti del programma.

STRUTTURA CONDIZIONALE

Le condizioni che il COMMODORE PLUS-4 puo' valutare, sono relazioni del tipo:

1>2
2<8
A/3=SIN(Z)

Ad ogni espressione di questo tipo il calcolatore sa assegnare il valore VERO o FALSO, esprimendo VERO con il numero -1 e FALSO con il numero 0. Prova ad esempio a scrivere PRINT 1>2 e premi RETURN: la risposta sara' 0, cioe' FALSO; ma se scrivi PRINT 2<8 e premi RETURN la risposta sara' -1 cioe' VERO. Puoi anche assegnare una risposta di questo tipo ad una variabile; ad esempio l'istruzione A=B=C vuole dire: poni A=-1 se B=C, altrimenti poni A=0. Una variabile usata come la variabile A prende il nome di VARIABILE BOOLEANA o LOGICA.
Quando parleremo di CONDIZIONI ci riferiremo quindi a relazioni del tipo A>2 o a variabili booleane.
Gli operatori relazionali del COMMODORE PLUS-4 sono 6:

= UGUALE
> MAGGIORE
< MINORE
<> DIVERSO
>= MAGGIORE O UGUALE
<= MINORE O UGUALE

Queste relazioni valgono anche per stringhe e costanti alfanumeriche:

"A"<"B" VERO (la lettera A viene prima della B)
"WA"<"RZ" FALSO (la lettera W viena dopo la R)

Piu' condizioni possono essere combinate per formare un'unica condizione con gli operatori logici AND, OR e NOT:
COND1 AND COND2: vero se sono vere entrambe
COND1 OR COND2: vero se sono vere o COND1 o COND2 o entrambe
NOT COND1: vero se COND1 e' falsa

ESEMPI:
A>0 AND A<4: VERO se 0<A<4
A<0 OR A>4: VERO se A non appartiene all'intervallo 0-4
NOT A: VERO se la variabile booleana A e' falsa

Se per combinare piu' condizioni usi piu' operatori logici, ricorda che il COMMODORE PLUS-4 esegue prima NOT poi AND e quindi OR. Se vuoi farle eseguire con un ordine diverso puoi usare le parentesi:

COND1 AND (COND2 OR COND3)

In questo caso eseguira' prima la OR e quindi la AND.

IF ... THEN ... ELSE
Se vuoi che il COMMODORE PLUS-4 esegua un'azione solo se e' verificata una condizione, devi usare l'istruzione IF... THEN.

La parola chiave IF deve essere seguita da una condizione, segui¬
ta dalla parola chiave THEN e da una o piu' istruzioni separate
da due punti. Tutte le istruzioni che appaiono dopo il THEN sul¬
la linea di programma vengono eseguite solo se la condizione e'
verificata:

Ecco alcuni esempi di istruzione IF:

```
IF A THEN COLOR1,7,3: PRINT "PIPPO"
```

oppure

```
IF X>0 AND X<10 THEN PRINT "0<X<10"
```

Nel primo caso A e' una variabile booleana, nel secondo X e' una
variabile reale.
Nel primo caso, se A e' vero (cioe' A=¬1) il COMMODORE PLUS-4
colora di blu' il cursore e scrive "PIPPO", se invece A e' falso
(cioe' A=0), non scrive nulla e non cambia il colore del curso¬
re.
Se A vale un numero diverso da 0 e da ¬1 il COMMODORE PLUS-4
considera A vera ma anche NOT A viene considerata vera.
Questo capita perche' il BASIC non possiede delle vere e proprie
variabili booleane (i cui valori siano cioe' solo VERO o FALSO),
ma permette che il programmatore usi le variabili numeriche co-
me booleane.
Assegna quindi i valori 0 e ¬1 (intendendo FALSO e VERO) alle
variabili numeriche usate come booleane, e intende FALSE le
variabili numeriche che valgono 0 e VERE le variabili numeriche
che hanno un valore diverso da 0. L'operatore NOT e' un'ope¬
ratore logico che puo' essere applicato anche su numeri interi e
che per 0 vale ¬1 ma per 5, ad esempio, vale -6: succede quindi
che 5 e NOT 5 siano entrambi VERI (sono entrambi diversi da 0).
Nel BASIC del COMMODORE PLUS¬4, applicando NOT a un numero reale
R, come risultato si ottiene NOT INT (R). I numeri reali per i
quali NOT R = 0 sono quindi quelli compresi tra 0 (escluso) e ¬1.
Si puo' completare una frase IF con la parola chiave ELSE: deve
essere posta dopo l'istruzione (o le istruzioni) che segue THEN,
preceduta da due punti e seguita dalla o dalle istruzioni che si
vogliono far eseguire se la condizione non e' verificata: ad
esempio

```
500 IF A>2 THEN GOSUB 1000: ELSE GOSUB 2000
510 ...
```

Se A>2 esegue la routine in 1000 e dopo prosegue da 510.
Se A<=2 esegue la routine in 2000 e dopo prosegue da 510.

I CICLI

E' possibile far eseguire al tuo COMMODORE PLUS-4 una o piu' azioni infinite volte, come in ES2.6:

```
1 REM ES2.6
10 PRINT"CIAO"
20 GOTO10
```

Ma possiamo voler far ripetere al calcolatore un'azione, o piu' azioni, per un numero prefissato di volte o fino a che una certa condizione venga verificata: per far cio' usereremo le istruzioni BASIC
FOR ... NEXT e
DO ... LOOP

I CICLI FOR ... NEXT
Le istruzioni FOR ... NEXT servono a far eseguire al calcolatore una o piu' azioni per un numero prefissato di volte. Esse debbono essere poste rispettivamente all'inizio e alla fine della serie di istruzioni che si vuole far ripetere.
La FRASE FOR puo' essere di questo tipo:

FOR I=1 TO 10

La lettera I e' il nome di una variabile di controllo, e puo' essere sostituita da un qualunque altro nome valido per una variabile numerica, i numeri 1 e 10 sono il valore iniziale e finale della variabile: questo ciclo viene quindi ripetuto 10 volte e la variabile I assume i valori da 1 a 10.
La frase che serve per chiudere la serie di azioni che vogliamo far eseguire per dieci volte e' NEXT I o, piu' semplicemente NEXT.
All'uscita dal ciclo la variabile di controllo contiene l'ultimo valore raggiunto, quello per il quale il ciclo non e' stato eseguito.
Se, per esempio, vogliamo far scrivere al COMMODORE PLUS-4 i numeri da 1 a 10, dobbiamo programmarlo nel seguente modo:

```
1 REM ES2.7
10 FOR I=1 TO 10
20 PRINT I
30 NEXT I
```

Il COMMODORE PLUS-4, quando trova il comando FOR capisce che deve iniziare un ciclo la cui variabile di controllo e' I; ad essa assegna il valore iniziale (nel nostro caso 1) e passa quindi ad eseguire la prossima istruzione. Quando incontra NEXT aggiunge

alla variabile di controllo 1 e, se I e' minore o uguale al valore finale torna all'istruzione che segue la frase FOR (nel nostro caso PRINT I), altrimenti prosegue. La frase FOR puo' essere completata dalla parola chiave STEP: questa, se usata, deve essere seguita da un numero reale, positivo o negativo, che viene sommato alla variabile di controllo quando il COMMODORE PLUS-4 trova l'istruzione NEXT. Questo numero si chiama INCREMENTO. Se l'incremento e' negativo il numero iniziale deve essere maggiore del numero finale e, arrivato all'istruzione NEXT, il calcolatore controlla che I sia ancora MAGGIORE o uguale al valore finale per tornare al ciclo.
Per capire meglio abbiamo preparato il programma ES2.8:

```
1 REM ES2.8
10 FOR I=10 TO 9 STEP -0.6
20 PRINT I
30 NEXT I
40 PRINT "FINE"
```

Cerchiamo di capire passo per passo che operazioni compie il COMMODORE PLUS-4 quando esegue ES2.8, commentando le linee:

. 10: inizializza il ciclo e pone la variabile I=10
. 20: scrive il valore di I, cioe' 10
. 30: incrementa la variabile di controllo di -0.6 cioe' I=10+(-0.6)=9.4 e poiche' I e' maggiore di 9 (valore finale) torna alla linea 20
. 20: scrive il valore di I, cioe' 9.4
. 30: incrementa la variabile di controllo di -0.6 cioe' I=9.4+(-0.6)=8.8 e poiche' I e' minore di 9 (valore finale) prosegue
. 40: scrive FINE e termina.

Se il valore iniziale e' maggiore di quello finale e l'incremento e' positivo (o se il valore iniziale e' minore di quello finale e l'incremento e' negativo), le istruzioni comprese tra FOR e NEXT vengono eseguite una volta.

Si possono far compiere piu' cicli FOR ... NEXT posti uno dentro l'altro come in ES2.9

```
1 REM ES2.9
10 FOR I=1 TO 10
20 FOR J=1 TO 30
30 NEXT J
40 NEXT I
```

Si possono "inscatolare" al massimo 10 cicli uno dentro all'altro. Se si tenta di inserire l'undicesimo, quando incontra

l'inizializzazione di questo, il COMMODORE PLUS-4 da' il messaggio di errore OUT OF MEMORY, poiche' ha gia' occupato tutta l'area in cui puo' memorizzare il numero della prima linea del ciclo, il valore finale della variabile di controllo e l'incremento.
E' sbagliato, invece, far eseguire dei cicli concatenati come in ES2.10

```
1 REM ES2.10
10 FOR I=1 TO 10
20 FOR J=1 TO 30
30 NEXT I
40 NEXT J
```

In un caso come questo, se dopo l'istruzione NEXT e' indicata la variabile a cui si riferisce, il COMMODORE PLUS-4 da' il messaggio di errore NEXT WITHOUT FOR; altrimenti eseguira' i due cicli come se fossero "inscatolati" nel modo corretto. Ti consigliamo quindi di scrivere sempre il nome della variabile di controllo a cui si riferisce un NEXT: in questo modo se hai inanellato male due cicli FOR ... NEXT il COMMODORE PLUS-4 ti segnala l'errore invece di comportarsi in maniera diversa da come tu pensavi. Un altro buon motivo per scrivere la variabile di controllo dopo un NEXT e' la LEGGIBILITA' del programma: se cioe' cerchi di capire cosa volevi far fare al tuo COMMODORE PLUS-4 con un programma scritto tre mesi fa', ti puo' essere di grande aiuto aver scritto, dopo ogni NEXT la variabile a cui ti riferivi, anche se al momento ti sembrava inutile.

Per chiudere due cicli "inscatolati" correttamente puoi usare anche la forma NEXT I,J che e' perfettemente equivalente a NEXT I: NEXT J.

Un'ultima osservazione su questo genere di cicli: il valore iniziale, il valore finale e l'incremento possono essere numeri interi o reali, variabili, o espressioni matematiche del tipo:

```
1 REM ES2.11
10 FOR I=LOG(A+3) TO EXP(8) STEP SIN(7.4)
```

I CICLI DO ... LOOP
Le istruzioni DO ... LOOP permettono di far eseguire al COMMODORE PLUS-4 una o piu' azioni, fino a che una condizione non venga verificata.
Se usate come in ES2.12, queste istruzioni provocano la ripetizione infinita delle istruzioni comprese tra DO e LOOP.

```
1 REM ES2.12
10 DO
20 PRINT "CIAO"
30 LOOP
```

Le istruzioni che servono per uscire dal ciclo sono UNTIL, WHILE e EXIT.

UNTIL deve essere usato dopo DO o dopo LOOP e deve essere seguito da una condizione, come esemplificato in ES2.13.a e in ES2.13.b.

```
1 REM ES2.13.A
10 DO UNTIL A$<>""
20 GET A$
30 LOOP
```

```
1 REM ES2.13.B
10 DO
20 GET A$
30 LOOP UNTIL A$<>""
```

Se la condizione A$<>"" non viene verificata, il COMMODORE PLUS-4 continua a ciclare come se non ci fosse UNTIL; ma appena la variabile A$ contiene qualche cosa (perche' e' stato premuto un tasto), il COMMODORE PLUS-4 esce dal ciclo e si ferma. Con UNTIL, quindi il calcolatore cicla FINO A CHE la condizione non sia verificata. La differenza tra il programma ES2.13.a e il programma ES2.13.b e' che, mettendo UNTIL dopo LOOP, le istruzioni tra DO e LOOP vengono eseguite almeno una volta (come nei cicli FOR ... NEXT); usando invece UNTIL dopo DO, le istruzioni tra DO e LOOP possono non essere eseguite neanche una volta (se si entra nel ciclo quando la condizione e' gia' verificata).
Anche WHILE deve essere usato dopo DO o dopo LOOP e, come UNTIL, deve essere seguito da una condizione. A differenza di UNTIL il calcolatore cicla MENTRE la condizione che segue WHILE e' vera, cioe' fino a quando la condizione diventa falsa. In altre parole WHILE COND e' come dire UNTIL NOT COND.
L'ultima istruzione che serve per uscire dai cicli DO LOOP e' EXIT. Puo' essere messa come una qualunque istruzione tra DO e LOOP e, quando viene trovata, l'esecuzione del programma salta all'istruzione dopo LOOP, cioe' esce dal ciclo. Conviene quindi usare l'istruzione EXIT cosi':

IF COND THEN EXIT

Il suo effetto e' cosi quello di UNTIL con la differenza che puo' essere messo in un qualunque punto del ciclo: in questo modo il

ciclo puo' essere abbandonato dopo aver eseguito solo una parte delle istruzioni che lo compongono.

Valgono le stesse norme di nidificamento ("inscatolamento") dei cicli FOR ... NEXT.
Puoi usare al massimo 39 cicli DO ... LOOP "inscatolati".
Nell'Appendice A in Figura A.2 sono riportati gli schemi delle istruzioni per il controllo dei cicli.

I SOTTOPROGRAMMI

Si puo' usare il nome SOTTOPROGRAMMA o SUBROUTINE, con lo stesso significato.
Quando il calcolatore trova l'istruzione GOTO N, va semplicemente a eseguire le istruzioni che trova nella linea N. Se invece trova l'istruzione GOSUB N, prima di saltare alla linea N, memorizza il punto del programma in cui si trova (o memorizza che l'istruzione e' stata data in modo diretto). Quando il COMMODORE PLUS-4 trova l'istruzione RETURN torna all'istruzione immediatamente successiva all'istruzione GOSUB. Questo fatto e' molto utile nel caso in cui devi far eseguire molto spesso un gruppo di istruzioni. Ad esempio, se devi sistemare i colori dello schermo in piu' punti puoi organizzare il tuo programma come segue.

```
10 ........
20 ........
..
..
100 GOSUB 1000
110 .......
...
...
230 GOSUB 1000
240 .......
...
...
280 GOSUB 1000
...
...
400 END

1000 COLOR 0,1
1010 COLOR 1,7,4
1020 COLOR 4,1
1030 RETURN
```

L'insieme di istruzioni dalla linea 1000 alla linea 1030 prende il nome di SUBROUTINE o SOTTOPROGRAMMA. Le SUBROUTINE sono molto

utili anche nei casi in cui devi far eseguire molte istruzioni al calcolatore solo se una condizione e' verificata, come in ES2.14.

```
1 REM ES2.14
10 DO
20 INPUT"COME TI CHIAMI ";A$
30 IFA$="FINE"THEN END
40 IFA$="PIPPO"THENGOSUB1000:ELSEGOSUB2000
50 LOOP
1000 COLOR0,7,6
1020 COLOR1,2,7
1030 COLOR4,7,6
1040 PRINT"◣CIAO "A$
1050 PRINT"SONO MOLTO CONTENTO DI VEDERTI"
1060 RETURN
2000 COLOR0,1
2010 COLOR1,5,3
2020 COLOR4,1
2040 PRINT"◣VAI VIA "A$
2050 PRINT"OGGI SONO DI MALUMORE"
2060 RETURN
```

In questo programma le ROUTINE sono 2:
.La prima, dalla linea 1000 alla linea 1060, viene eseguita se la stringa ricevuta nella linea 20 e' "PIPPO".
.La seconda, dalla linea 2000 alla linea 2060, viene eseguita se la stringa ricevuta nella linea 20 non e' "PIPPO".

Come per i cicli, puoi nidificare anche le ROUTINE; e come per i cicli DO ... LOOP, il numero massimo e' di 39 ROUTINE "inscatolate". L'area usata per memorizzare le linee a cui deve tornare l'esecuzione del programma dopo una SUBROUTINE e' la stessa che viene utilizzata per i cicli FOR ... NEXT e per memorizzare la prima linea dei cicli DO ... LOOP: quest'area di memoria si chiama STACK.
Il numero massimo di SUBROUTINE nidificabili e' dunque 39, se il piu' interno di queste non contiene a sua volta uno o piu' cicli FOR ... NEXT o DO ... LOOP. Il decimo ciclo FOR ... NEXT puo' contenere ancora 3 tra GOSUB e DO ... LOOP nidificati.

I SALTI CALCOLATI

L'istruzione GOTO provoca un salto senza condizioni in un determinato punto di un programma. L'istruzione IF...THEN...ELSE provoca salti sotto condizione. Esistono due istruzioni:
ON...GOTO e ON...GOSUB
che provocano salti in base a un calcolo, cioe' al valore che ha al momento dell'esecuzione un'espressione numerica, che in particolare puo' essere solo una variabile.
Esse si scrivono:

ON esp GOTO n1,n2,n3,...,ni

ON esp GOSUB n1,n2,n3,...,ni

e agiscono cosi':
se l'espressione vale 1 il salto avviene a n1, primo numero di linea citato dopo GOTO o GOSUB, se essa vale 2 al secondo, se essa vale i all'i-esimo. Se il valore dell'espressione risulta 0 o supera il numero dei numeri di linea presenti, il programma prosegue dalla linea seguente. Se il valore risulta negativo si ottiene un messaggio di errore.
Nel caso di GOSUB viene eseguito il sottoprogramma che inizia in ni e al RETURN viene eseguita l'istruzione successiva a ON...GOSUB.
Vediamo un esempio:

```
1 REM ES2.15
10 PRINT"     SCELTA PROCEDURA"
15 PRINT"   1: CALCOLO":PRINT"   2: VERIFICA"
20 PRINT"   3: STAMPA":PRINT"   9: FINE"
30 GETKEYA$:IFA$<"1"ORA$>"3"ANDA$<>"9"THEN30
35 ON VAL(A$) GOSUB 100,200,300,900
100 PRINT100:STOP
200 PRINT200:STOP
300 PRINT300:STOP
900 PRINT900:STOP
```

Nel programma ES2.15 viene proposto un menu' di scelta di procedure; viene ricevuta la risposta in A$ con GETKEY, essa viene controllata e accettata solo se corretta. Alla linea 35 viene scelto il sottoprogramma da eseguire in base al valore di A$. Alle linee 100, 200, 300 e 900 viene stampato un numero corrispondente al numero di linea e si ha uno STOP. Sarebbe poco ortodosso terminare un sottoprogramma con STOP, ma l'esempio vuole solo mettere in evidenza l'istruzione ON...GOSUB.

2.8 PROGRAMMARE LA GRAFICA

Ci occupiamo qui delle possibilita' grafiche del COMMODORE PLUS-4.

ALTA RISOLUZIONE E MULTICOLORE.

Al momento dell'accensione, il video puo' mostrare caratteri alfanumerici e grafici disposti in 1000 possibili CASELLE. Ogni casella e' divisa a sua volta in 64 CASELLINE disposte su 8 righe di 8 elementi ciascuna. Le caselline prendono il nome di PIXEL. Se in una casella coloriamo opportunamente alcuni pixel, possiamo ottenere il disegno di una lettera o di un numero o di un simbolo grafico. Ad esempio, possiamo ottenere la lettera "A" in questo modo:

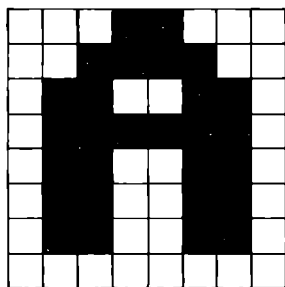



Figura 2.3 Immagine lettera A

I pixel contenuti nel video del COMMODORE PLUS-4 sono quindi 64*1000=64000 disposti su 25*8=200 righe di 40*8=320 elementi. Quando accendi il calcolatore non puoi controllare i pixel individualmente: non puoi decidere se un singolo pixel deve essere del colore dello sfondo o no; puoi solo far visualizzare i simboli (alfanumerici o grafici) dei due set di caratteri del COMMODORE PLUS-4 nelle 1000 caselle del video. Chiamiamo questo modo di visualizzazione MODO TESTO. Oltre al modo testo esistono il MODO ALTA RISOLUZIONE e il MODO MULTICOLORE. Nel primo puoi controllare ogni pixel del video, nel secondo controlli solamente 160 pixel per riga (ogni pixel e' largo come due pixel in alta risoluzione), ma puoi gestire meglio il colore.

IL PENNINO (O CURSORE GRAFICO).

Quando disegni su un foglio di carta, sposti la penna a volte premendola sul foglio, altre tenendola sollevata. Anche il tuo COMMODORE PLUS-4 usa un immaginario pennino per disegnare. All'accensione o dopo un'istruzione di pulizia dello schermo, il pennino e' posizionato nell'angolo in alto a sinistra dello schermo (il punto 0,0). Puoi, con l'istruzione LOCATE X,Y, posizionare il pennino nell'X-esimo punto dell'Y-esima riga, tenendolo SOLLEVATO (cioe' senza lasciare la traccia). Tracciando rette, circonferenze, poligoni o altre figure il pennino viene spostato e viene lasciato sull'ultimo punto disegnato. Il pennino serve come punto sottointeso in alcune istruzioni (ad esempio si puo' tracciare una retta dal pennino ad una altro punto o disegnare un rettangolo che abbia come vertice il punto occupato dal pennino); inoltre il pennino puo' essere usato come origine di un sistema di riferimento di coordinate relative.

SISTEMI DI COORDINATE

Il modo piu' spontaneo di indicare un punto dello schermo e' quello di dire a quale riga e quale colonna appartiene: il punto in alto a destra e', per esempio, il punto della colonna 319, riga 0 (piu' brevemente punto 319,0). E' proprio con la coppia di numeri 319,0 che ci si riferisce, normalmente, al punto in alto a destra. Questo modo e' sicuramente molto immediato, ma puo' essere utile, a volte, indicare un punto RELATIVAMENTE al pennino. Puo' essere utile ad esempio voler indicare il punto che sta sopra al pennino di 7 punti: per far cio' basta indicare il punto +0,-7. Il COMMODORE PLUS-4, infatti, capisce che vuoi indicare un punto relativamente al cursore grafico, se poni un segno + o - prima delle coordinate: un + prima del numero di colonna indica un punto piu' a destra del pennino; un + prima del numero di riga indica un punto piu' in basso del pennino. E possibile anche indicare la colonna di un punto in maniera assoluta e la riga in maniera relativa (o viceversa). Ad esempio il punto 5,-9 e' il punto della colonna 5 che appartiene alla nona riga sopra il cursore grafico. Un altro modo di indicare un punto relativamente al pennino e quello di dare una distanza e un angolo, cioe' di indicare il punto usando un sistema di COORDINATE POLARI. Per far cio' devi separare le due coordinate con un punto e virgola (anziche' una virgola). L'angolo deve essere misurato in gradi, in senso orario partendo dalla semiretta dal pennino verso l'alto. Ad esempio il punto 30;90 e' trenta punti piu' a destra del pennino.

MODI GRAFICI (L'ISTRUZIONE GRAPHIC)

Nel modo alta risoluzione e multicolore il calcolatore usa una zona di memoria, diversa da quella che usa in modo testo, per ricordare il contenuto del video. E' possibile, quindi, passare da un modo di visualizzazione all'altro senza perdere il contenuto dei quadri video di testo o di grafica. L'istruzione che permette di passare da un modo di visualizzazione all'altro e' GRAPHIC:

GRAPHIC 0 visualizza in modo testo.

GRAPHIC 1 visualizza in modo alta risoluzione.

GRAPHIC 2 visualizza i 4/5 in alto in modo alta risoluzione e il quinto rimanente in modo testo.

GRAPHIC 3 visualizza in modo multicolore.

GRAPHIC 4 visualizza i 4/5 in alto in modo multicolore e il quinto rimanente in modo testo.

Aggiungendo ",1" a questi comandi ottieni la pulizia dello schermo (grafico se passi a un modo grafico, testo se passi al modo testo, entrambi se passi a un modo misto). Puoi pulire lo schermo senza cambiare modo grafico con l'istruzione SCNCLR.
Se non usi mai un modo di visualizzazione grafico, il COMMODORE PLUS-4 non usa memoria per la pagina grafica; quando pero' entri in modo grafico il calcolatore deve "rubare" 12288 byte di memoria al BASIC, che rimane con 48381 byte (vedi figura 2.4). Se torni al modo testo con l'istruzione GRAPHIC 0, i byte rubati non vengono restituiti al BASIC; vengono pero' restituiti tornando al modo testo con l'istruzione GRAPHIC CLR.

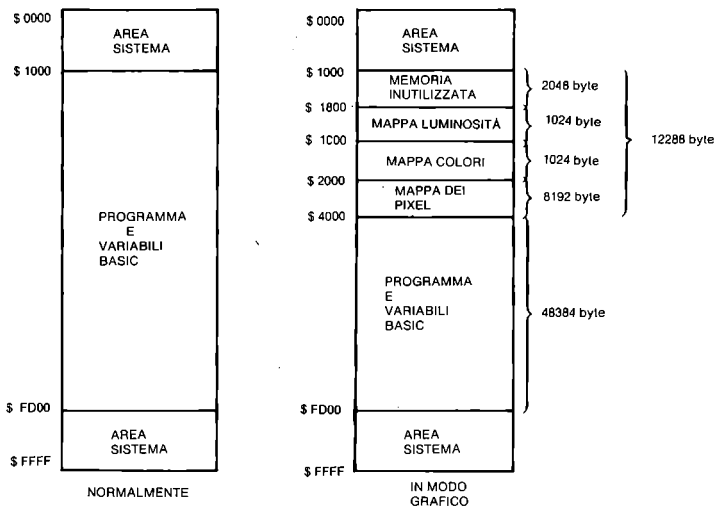



Figura 2.4 Uso della memoria nel modo grafico

I COLORI NEI MODI GRAFICI

Abbiamo gia' visto la funzione dell'istruzione COLOR, ma non abbiamo chiarito quali sono le zone 2 e 3. In modo alta risoluzione e' consentito disegnare nel colore dello sfondo (0) o nel

colore dei caratteri (1); in modo multicolore, invece, hai a disposizione altri due colori, che corrispondono alle zone 2 e 3. Chiamiamo i colori 1, 2, 3: inchiostro 1, inchiostro 2 e inchiostro 3.
Se, con l'istruzione COLOR, cambi il colore dell'inchiostro 3, tutti i disegni gia' fatti sul video con l'inchiostro 3, cambiano immediatamente colore. Se invece cambi il colore dell'inchiostro 2 (o 1), solo i disegni che d'ora in poi farai con l'inchiostro 2 (o 1), avranno un altro colore.

LE ISTRUZIONI GRAFICHE

Nel seguito descriveremo una serie di funzioni che agiscono sulla pagina grafica. Tutte queste funzioni saranno seguite da una serie di parametri, il primo dei quali e' il colore (0=colore di sfondo, 1=inchiostro 1, 2=inchiostro 2, 3=inchiostro 3). Fanno eccezione a questa regola le istruzioni SSHAPE e GSHAPE. Spesso si possono sottointendere alcuni parametri: per fare questo basta non scrivere il parametro, lasciando la punteggiatura richiesta. Quando il parametro sottointeso e' un punto devi sottointendere le coordinate e il separatore. Se usi istruzioni grafiche mentre sei in modo testo e non e' occupata la memoria per la pagina grafica (cioe' non sei mai entrato in modo grafico da quando hai acceso il calcolatore oppure hai usato l'istruzione GRAPHICCLR), il COMMODORE PLUS-4 ti risponde con il messaggio di errore
?NO GRAPHICS AREA ERROR
Se, invece, la memoria per la pagina grafica e' stata occupata, le istruzioni grafiche, anche se date in modo testo, vengono eseguite ugualmente sulla pagina grafica che viene considerata ad alta risoluzione (solo che tu non ne vedi l'effetto subito): se usi, infatti, il colore 2 o 3 (propri del modo multicolore) il COMMODORE PLUS-4 ti risponde con il messaggio di errore
?ILLEGAL QUANTITY ERROR
come quando e' in modo alta risoluzione. Questi messaggi di errore non vengono dati se l'istruzione e' CHAR.

PUNTI E RETTE (L'ISTRUZIONE DRAW)

Consideriamo ora una delle piu' versatili istruzioni grafiche del COMMODORE PLUS-4: DRAW. Questa istruzione consente di disegnare o cancellare punti, rette e linee spezzate. Nella forma completa l'istruzione deve essere seguita dal colore, le coordinate di un punto, la parola chiave TO e le coordinate di un secondo punto: l'istruzione traccia una linea del colore indicato dal primo al secondo punto. Ad esempio:
DRAW 2,0,0 TO 100,100
disegna una linea del colore dell'inchiostro 2, da 0,0 a 100,100
DRAW 0,0,0 TO 100,100

cancella la linea appena disegnata (disegnandone una del colore dello sfondo).
Omettendo il primo parametro il COMMODORE PLUS-4 assume il colore uguale all'inchiostro 1:
DRAW,12;100 TO +50,-20
disegna una retta di colore 1.
Omettendo il primo punto il COMMODORE PLUS-4 traccia una retta dal cursore grafico al punto indicato. Poiche' il pennino si trova, dopo un DRAW, sul secondo punto, le istruzioni
LOCATE0,0:DRAW TO 10,10:DRAW TO 10,20:DRAW TO 20,10
disegnano una linea spezzata di colore 1 i cui vertici sono i punti indicati nelle istruzioni.
Se, infine, ometti il secondo punto il COMMODORE PLUS-4 disegna solo il primo punto (ove lascia il pennino).

CIRCONFERENZE, ELLISSI E POLIGONI (L'ISTRUZIONE CIRCLE)

Il modo piu' semplice per usare CIRCLE e' quello in cui indichi solo il colore, le coordinate del centro, la lunghezza del raggio.
CIRCLE2,100,100,30
disegna una circonferenza di colore 2 centrata in 100,100 di raggio 30.
Sottointendendo il colore ottieni il colore dell'inchiostro 1, sottointendendo il centro la circonferenza viene centrata sul pennino; il raggio non puo' essere sottointeso.
Gli altri parametri che possono seguire sono:
RAGGIO Y: e' il semiasse verticale di un'ellissi il cui semiasse orizzontale sia lungo quanto il raggio. Se non c'e' viene considerato uguale al raggio X. In modo multicolore devi ricordare che un pixel in orizzontale ne vale due in verticale.
ANGOLO DI PARTENZA: normalmente il calcolatore comincia a disegnare la circonferenza dal punto a 0 gradi dalla semiretta verso l'alto uscente dal centro e la finisce nel punto a 360 gradi (dove viene lasciato il pennino). Se viene dato questo parametro il COMMODORE PLUS-4 comincera' a disegnare la circonferenza da quella posizione.
ANGOLO FINALE: l'angolo raggiunto il quale il calcolatore smette di disegnare la circonferenza. Grazie a questo e al precedente parametro puoi disegnare archi di circonferenza.
ROTAZIONE: l' angolo (in senso orario) di cui viene ruotata la figura:
CIRCLE 1,100,100,60,30,,,45
disegna una ellissi il cui asse maggiore e' ruotato di 45 gradi.
Questa operazione viene eseguita male in modo multicolore: infatti il calcolatore non tiene conto della forma asimmetrica dei pixel in questo modo, per cui la figura risulta allungata.

LATI: volendo disegnare un poligono regolare di N lati devi usare come ultimo parametro 360/N:
CIRCLE,100,100,50,,,,,360/6
disegna un esagono regolare.

L'ISTRUZIONE PAINT

Se vuoi dipingere un'area, ti basta dare al COMMODORE PLUS-4 l'istruzione PAINT seguita dal colore che vuoi usare e dalle coordinate di un punto contenuto nell'area che vuoi dipingere
CIRCLE,100,100,50:PAINT,100,100
disegna un cerchio pieno.
Se sottointendi il punto che individua l'area, il calcolatore parte a dipingere dal pennino.
Dopo PAINT il pennino viene lasciato sul punto che individua l'area (punto iniziale).
Puoi aggiungere un ultimo parametro: se vale 1 il calcolatore considera che l'area e' delimitata da un qualunque colore diverso dallo sfondo. Se e' 0 il colore che delimita l'area e' solo il colore 1.

DISEGNARE RETTANGOLI (L'ISTRUZIONE BOX)

Con l'istruzione BOX puoi disegnare un rettangolo semplicemente dando il colore e due vertici opposti del rettangolo. Il colore sottointeso e' il colore dell'inchiostro 1, il secondo angolo, se sottointeso e' il pennino.
Dopo questa istruzione il pennino viene lasciato sul secondo angolo.
Possono seguire altri due parametri: il primo indica la rotazione (in senso orario, espressa in gradi). Se il secondo e' 1 il rettangolo viene dipinto.

SCRIVERE CARATTERI IN MODO GRAFICO (L'ISTRUZIONE CHAR)

L'istruzione CHAR e' un'istruzione che vale sia in modo grafico che in modo testo. Scrive su un video piuttosto che l'altro a seconda del modo grafico in cui si trova il calcolatore. Nei modi MISTI agisce solo sulla pagina grafica.
Con questa istruzione puoi scrivere tutte le parole che vuoi nella posizione che preferisci.
CHAR1,10,10,"PIPPO"
scrive PIPPO in colore 1 partendo dalla colonna 10, riga 10 in modo testo, colonna 80, riga 80 in modo alta risoluzione. In modo multicolore i risultati sono poco apprezzabili.
Puoi aggiungere a CHAR un ulteriore parametro: se e' 1 la scritta viene stampata in campo inverso.
L'istruzione CHAR accetta anche una serie di CHR$ (separati da +)

come stringa. In modo testo vengono stampati normalmente, in modo grafico i caratteri di controllo vengono scritti come se fossero tra virgolette, ma in campo diretto. Il CHR$(255) non e' π ma un simbolo grafico.

TRASFERIRE PARTI DI SCHERMO GRAFICO (LE ISTRUZIONI SSHAPE E GSHAPE)

Il COMMODORE PLUS-4 ti da' la possibilita' di memorizzare una parte rettangolare della pagina grafica in una variabile stringa. Per fare cio' devi usare l'istruzione SSHAPE: tale istruzione deve essere seguita dal nome della variabile in cui si vuole salvare la zona e dai vertici che delimitano il rettangolo (come in BOX).
Per porre questa sezione in un altro punto della pagina grafica devi usare l'istruzione GSHAPE seguita dal nome della variabile che contiene la zona salvata con SSHAPE, e le coordinate in cui vuoi porre l'angolo in alto a sinistra del rettangolo.
La grandezza dell' area che puoi memorizzare e' limitata dalla lunghezza massima delle stringhe: 255 caratteri. Se vuoi calcolare la lunghezza della stringa che conterra' l'area devi usare le formule:

in alta risoluzione:
```
INT((ABS(X1-X2)+1)/8+.99)*(ABS(Y1-Y2)+1)+4
```

in multicolore:
```
INT((ABS(X1-X2)+1)/4+.99)*(ABS(Y1-Y2)+1)+4
```

Dove X1,Y1,X2,Y2 sono rispettivamente la prima e la seconda coordinata del primo angolo e la prima e la seconda coordinata del secondo angolo.

GSHAPE puo' essere seguita da un ultimo parametro: il modo di porre sul video la zona salvata:
- 0 pone la figura come e',
- 1 la pone in campo inverso,
- 2 la pone facendo la OR con l'area,
- 3 la pone facendo la AND con l'area,
- 4 la pone facendo la XOR con l'area.

I modi 2, 3 e 4 possono risultare poco chiari.

Vediamo gli effetti su una pagina ALTA RISOLUZIONE:
- modo 2, rimane lo sfondo sulla parte di colore 0 della figura,
- modo 3, disegna solo le zone in cui sia lo sfondo che la figura sono di colore 1,
- modo 4, disegna in colore 1 su sfondo di colore 0 e in colore 0 su sfondo di colore 1.

E gli effetti su pagina MULTICOLORE:
modo 1 si scambiano colore le parti di colore 0 e 3 e quelle di colore 1 e 2,
modi 2, 3 e 4:

CS	CF	M2	M3	M4
0	0	0	0	0
0	1	1	0	1
0	2	2	0	2
0	3	3	0	3
1	0	1	0	1
1	1	1	1	0
1	2	3	0	3
1	3	3	1	2
2	0	2	0	2
2	1	3	0	3
2	2	2	2	0
2	3	3	2	1
3	0	3	0	3
3	1	3	1	2
3	2	3	2	1
3	3	3	3	0

(se CS e' il colore dello sfondo in un punto e CF e' il colore della figura nello stesso punto, allora M2 e' il colore che viene visualizzato in quel punto ponendo l'immagine sullo sfondo col modo 2, M3 col modo 3 e M4 col modo 4).

FUNZIONI GRAFICHE

Il COMMODORE PLUS-4 possiede 4 funzioni riguardanti grafica e colore: RGR, RCLR, RLUM, RDOT.

RGR, il cui argomento puo' essere qualunque, torna il modo grafico in cui e' il calcolatore (ricordiamo che il modo grafico si cambia con l'istruzione GRAPHIC).

RCLR torna il colore della zona che viene passata come parametro alla funzione:
RCLR (0) torna il colore dello sfondo,
RCLR (1) torna il colore dell'inchiostro 1,
RCLR (2) torna il colore dell'inchiostro 2,
RCLR (3) torna il colore dell'inchiostro 3,
RCLR (4) torna il colore del bordo.

RLUM torna la luminosita' della zona che viene passata come parametro alla funzione:

RLUM (0) torna la luminosita' dello sfondo,
RLUM (1) torna la luminosita' dell'inchiostro 1,
RLUM (2) torna la luminosita' dell'inchiostro 2,
RLUM (3) torna la luminosita' dell'inchiostro 3,
RLUM (4) torna la luminosita' del bordo.

RDOT fornisce informazioni sul pennino:
RDOT (0) torna la colonna su cui e' posizionato il pennino,
RDOT (1) torna la riga su cui e' posizionato il pennino,
RDOT (2) torna il colore del punto su cui si trova il pennino.

Ecco il listato di un programma che usa la funzione RDOT.

```
1 REM ES2.16
5 COLOR0,1:COLOR4,1:YY=1:NC=40:C=8:TRAP500
10 GRAPHIC1,1
20 GETKEYA$
30 IFA$=CHR$(13)THENXX=0:YY=YY+1:GOTO20
40 IFA$=CHR$(20)THENGOSUB300:GOTO20
50 IFA$=CHR$(27)THENGOSUB400:GOTO20
60 GOSUB200
70 XX=XX+C:IFXX>319-CTHENXX=0:YY=YY+1
80 GOTO20
200 COLOR1,1:CHAR,0,0,A$:COLOR1,7,7
210 FORX=0TOC-1
220 FORY=0TO7
230 LOCATEX*8/C,Y
240 DRAWRDOT(2),XX+X,YY*8+Y
250 NEXTY,X
260 RETURN
300 XX=XX-C:YY=YY+(XX<0):XX=XX-320*(XX<0)
310 A$=" ":GOSUB200:RETURN
400 GETKEYA$,B$:NC=VAL(A$)*10+VAL(B$)
410 C=320/NC:RETURN
500 FORX=0TO7:POKE(8192+X),0:NEXTX:GRAPHIC0:SCNC
LR
```

Lo scopo di questo programma e' quello di scrivere sulla pagina grafica caratteri di diversa larghezza con il seguente metodo: viene stampato, nell'angolo superiore sinistro, in nero, il carattere che si vuole allargare o rimpicciolire; la funzione RDOT sa comunque riconoscere se i punti che formano il carattere sono di inchiostro 1 o di sfondo; una routine riproduce quindi, punto per punto, in blu, nell'attuale posizione del cursore la lettera compressa o allargata. Per decidere quanti caratteri si vogliono visualizzare su ogni riga basta premere il tasto ESC seguito dal numero di caratteri per riga (due cifre). Il programma riconosce ed esegue DEL e RETURN.
Vediamo ora, linea per linea, come funziona il programma:

.0: schermo e bordo neri; riga sulla quale verra' stampato il prossimo carattere (YY) = 1; numero di caratteri per linea (NC) = 40; numero di pixel per ogni carattere (C) = 320/40 = 8. Per TRAP vedi Paragrafo 3.8. In questo caso manda l'esecuzione del programma alla linea 500 quando premi RUN/STOP.
.10: entra in alta risoluzione e pulisce lo schermo.
Linea 20: accetta un carattere da tastiera e lo pone nella variabile A$.
.30: se il tasto premuto e' RETURN (=CHR$(13)) va a capo: cioe' incrementa la riga e pone = 0 la colonna nella quale verra' stampato il prossimo carattere (XX indica il primo pixel da cui dovra' partire la stampa del prossimo carattere). Torna quindi alla linea 20 per chiedere il prossimo carattere.
.40: se il tasto premuto e' DEL (=CHR$(20)) esegue la subroutine in 300 che cancella il carattere precedente. Torna quindi alla linea 20.
.50: se il tasto premuto e' ESC (=CHR$(27)) esegue la subroutine in 400 che accetta un nuovo numero di colonne. Torna quindi alla linea 20.
.60: esegue la routine in 200 che stampa un carattere sullo schermo alta risoluzione.
.70: incrementa di C pixel il prossimo punto di stampa. Controlla che il prossimo carattere non ecceda la linea (XX>319+C): se cio' avviene va a capo.
.80: torna alla linea 20 per ricevere il prossimo carattere.

SUBROUTINE DI STAMPA

.200: scrive in alto a sinistra, in nero, il carattere che si vuole stampare e rimette blu il colore del cursore.
.210: inizializza il ciclo dei punti orizzontali del carattere (se il carattere e' largo 5 pixel il ciclo verra' ripetuto 5 volte).
.220: inizializza il ciclo dei punti verticali.
.230: pone il pennino sul punto del carattere che si vuole valutare.
.240: disegna un punto del colore che si legge sul carattere originale nel posto del carattere modificato.
.250: chiude i due cicli.
.260: fine della subroutine.

SUBROUTINE PER DEL

.300: decrementa l'indicatore di colonna di tanti pixel quanto e' la larghezza attuale di un carattere; se cosi' facendo il numero diventa minore di 0 viene decrementato il numero di linea e aggiunto 320 al numero di colonna.
.310: servendosi della subroutine in 200 stampa uno spazio che

cancella il carattere indesiderato.

SUBROUTINE PER ESC

.400: accetta due caratteri da tastiera e li considera un numero di due cifre: pone il numero di colonne (NC) pari al valore calcolato.
.410: pone la larghezza in pixel dei caratteri (C) = 320/NC.

SUBROUTINE PER STOP

.500: cancella la prima posizione carattere, torna in modo testo, pulisce lo schermo e il programma si arresta.

2.9 PROGRAMMARE L'ANIMAZIONE

Come per i cartoni animati, la televisione e il cinema, il movimento che vedi nei videogiochi (o comunque il movimento generato da un calcolatore) non e' vero movimento, ma una sequenza di figure statiche ognuna delle quali differisce di poco dalla precedente. Un grossolano esempio su come imitare un movimento col calcolatore e' dato dal programma ES2.17:

```
1 REM ES2.17
10 PRINTCHR$(147)
20 FORI=0TO38:CHAR,I,0," ●"
30 FORJ=1TO30
40 NEXTJ,I
50 RUN
```

Questo programma disegna, in uno schermo vuoto, una pallina, ogni volta una posizione piu' a destra della precedente. In questo modo sembra che la pallina corra verso destra. Ecco la spiegazione linea per linea del programma:

.10: pulisce lo schermo.
.20-40: per 39 volte stampa, ogni volta un carattere piu' a destra, uno spazio e una pallina. Lo spazio ha la funzione di cancellare la vecchia pallina. Il ciclo a vuoto di J serve a rallentare l'esecuzione. In questo modo visualizziamo, abbastanza rapidamente, una dopo l'altra delle figure che differiscono di poco dalle precedenti.
.50: ricomincia.

Il motivo per cui il risultato di questo programma non e' molto soddisfacente e' che la differenza tra un'immagine e l'altra e' troppo grande in rapporto alla grandezza dell'oggetto che si muove.

Proviamo ora a costruire un movimento che ci soddisfi di piu':

```
1 REM ES2.18
10 DIMA$(2):PRINTCHR$(147)
20 A$(0)="╭●╮":A$(1)="─●─":A$(2)="╰●╯"
30 FORI=0TO2:CHAR,0,0,A$(I)
40 FORJ=0TO90:NEXTJ,I:GOTO30
```

Con questo esempio imitiamo il battere delle ali di un uccellino. Il risultato, questa volta, e' un po' piu' bello perche' il movimento delle ali e' di qualche pixel, mentre prima la pallina si muoveva di un carattere alla volta. Vediamo come funziona il programma:

.10-20: memorizza le immagini necessarie nel vettore di stringhe A$ e pulisce lo schermo.
.30-40: stampa una dopo l'altra le tre immagini nello stesso posto. Il ciclo a vuoto di J rallenta, il programma prosegue fino a quando non premi STOP.

Per completare questo programma possiamo far VOLARE l'uccellino:

```
1 REM ES2.19
10 DIMA$(2):PRINTCHR$(147)
20 A$(0)="╭●╮":A$(1)="─●─":A$(2)="╰●╯"
30 Y=25:FORX=0TO24:Y=Y-1
40 GOSUB100
50 NEXTX
60 RUN
100 PRINTCHR$(147)
110 FORI=0TO2:CHAR,X,Y,A$(I)
120 FORJ=1TO50
130 NEXTJ,I
140 RETURN
```

Combinando il movimento delle ali a quello dell'uccellino si ottiene un risultato abbastanza buono. Il programma funziona cosi':

.10-20: come prima.
.30: posizione verticale dell'uccellino (Y) = 25 (piu' in basso possibile). Inizializza un ciclo da 0 a 24, la cui variabile indice e' X (posizione orizzontale), che decrementa la posizione verticale.
.40: fa volare l'uccellino da una casella alla casella immediatamente sopra a destra.
.50: chiude il ciclo.

.60: lancia nuovamente il programma.

ROUTINE DI MOVIMENTO

.100-140: svolgono la funzione del programma precedente con la differenza che la posizione in cui l'uccellino sbatte le ali e' data dai valori di X e Y.

2.10 PROGRAMMARE IL SUONO

Il COMMODORE PLUS-4 possiede due comandi diretti alla gestione del suono. Le possibilita' sonore del COMMODORE PLUS-4 sono abbastanza limitate, poiche' vi sono solamente due generatori sonori, e questi generatori producono solo suoni a onda quadra o a onda casuale (rumore bianco); inoltre non vi sono filtri ne' generatori di inviluppo. L'assenza di tutte queste potenzialita' e' forse un handicap, ma e' senza dubbio vantaggioso per quello che riguarda la facilita' d'uso.
I due comandi che gestiscono il suono sono:
VOL e SOUND.

Con essi e' assai semplice generare suoni ed effetti sonori.
VOL deve essere seguito da un numero compreso tra 0 e 8:
VOL 0 = volume a zero
VOL 8 = volume al massimo

Per generare un suono occorre anche specificarne la forma d'onda, la frequenza e la durata. Tutti questi parametri devono essere forniti all'istruzione SOUND.
SOUND V,F,D
produce una nota musicale col generatore V, la frequenza dipendente da F, e della durata di D sessantesimi di secondo.

- V e' un numero che puo' valere 1, 2 o 3. Se vale 1 il suono viene generato dal generatore 1, sempre con onda quadra, se vale 2 viene generato dal generatore 2 con onda quadra e se V=3 il suono viene sempre generato dal generatore 2 ma con onda casuale (rumore bianco). Ovviamente le voci 2 e 3 non possono suonare insieme, mentre possono suonare assieme voce 1 e voce 2 o voce 1 e voce 3.
- F e' un numero da cui dipende la frequenza, a cui e' legato dalla seguente espressione:
  F=1024-(111840.45/Hertz)
  Nell'Appendice H trovi i valori del parametro F per tutte le note musicali. Se hai orecchio puoi notare una certa stonatura nei suoni, dovuta al fatto che F e' un numero intero, che quindi e' generalmente approssimato.
- D e' un numero compreso tra 0 e 65535 che esprime la durata della nota in sessantesimi di secondo.

Segue il programma ES2.20 che suona le note contenute in linee DATA.

```
1 REM ES2.20
5 DIMMT(75)
10 RD=2↑(1/12)
20 F=7040
30 FORI=73TO1STEP-1
40 MT(I)=INT(1024.5-111840.45/F)
60 F=F/RD
70 NEXT
85 VOL8
90 DO
100 READA:IFA=0THENEXIT
110 SOUND1,MT(A),5:FORI=1TO100:NEXT
120 LOOP
9000 DATA17,15,17,13,17,12,17,10,17,8,17
9002 DATA6,17,15,13,15,15,13,15,12
9005 DATA15,10,15,8,15,6,15,5,15,13,12,13
9006 DATA13,12,13,10,13,8,13,6,13,5,13,4,13
9007 DATA12,10,12,12
9010 DATA10,12,9,12,7,12,5,12,3,12,1,12
9020 DATA10,8,10,0
```

COMMENTO A ES2.20

.0-70: viene definito il vettore MT che contiene tutti i valori di F in base al numero di nota da suonare. Partendo da una frequenza di 7040 Hertz calcoliamo il valore da porre nel vettore, e una per volta ricaviamo il valore della nota piu' bassa dividendo per RD che assume il valore della radice dodicesima di due. In questo programma la variabile F esprime la frequenza in Hertz della nota di cui vogliamo calcolare l'argomento per SOUND.

.90-120: ciclo in cui vengono letti i dati dalle linee DATA, e sono usati come indici del vettore MT. Il numero esprime la nota musicale secondo la seguente tabella:

DO	40	28	16	4
DO#	41	29	17	5
RE	42	30	18	6
RE#	43	31	19	7
MI	44	32	20	8
FA	45	33	21	9
FA#	46	34	22	10
SOL	47	35	23	11
SOL#	48	36	24	12
LA	49	37	25	13
LA#	50	38	26	14
SI	51	39	27	15
OTTAVA	4	3	2	1

Il vettore MT ha 76 elementi. Ti consigliamo di non usare gli elementi piu' alti, perche' rappresentano frequenze un po' troppo alte per il generatore di suoni.

CAPITOLO 3

OPERAZIONI AVANZATE IN BASIC

3.1 LE VARIABILI CON INDICE

Fino ad ora ci siamo occupati di variabili singole; nella programmazione risulta spesso utile poter riferire con lo stesso nome un GRUPPO DI VARIABILI. Sono disponibili per questo le VARIABILI CON INDICE, chiamate anche MATRICI o VETTORI (ARRAY in inglese). Si tratta di una struttura di dati nella quale si usa lo stesso nome per rappresentare un gruppo di variabili; per identificare un elemento del gruppo si usano uno o piu' numeri, detti indici, che forniscono la posizione di quell'elemento nel gruppo.
Vediamo un esempio: consideriamo un vettore di variabili di tipo numerico, di nome A. Il primo elemento di tale gruppo si chiama A(0), e ha le stesse proprieta' di ogni altra variabile di tipo numerico. Il secondo elemento si chiama A(1), e il suo valore e' completamente indipendente dal valore di A(0) e di tutti gli altri elementi del vettore. A(0) e A(1) sono due variabili diverse. Prova a eseguire le seguenti linee:

```
A(0)=10:A(1)=3.14
PRINT A(0);A(1)
```

Il calcolatore stampa 10 e 3.14. Un'osservazione, a questo punto, potrebbe essere questa: "Che cos'hanno di diverso i vettori dalle variabili normali? Anche la variabile A0 e' completamente separata da A1, e si risparmia di scrivere le parentesi!". La differenza c'e', ed e' importante. L'indice contenuto tra parentesi infatti puo' essere un numero o una variabile numerica, di cui viene considerata solo la parte intera:
K=1:PRINT A(K)
stampa ancora il valore di A(1), mentre:
K=1:PRINT AK
non stampa il valore della variabile A1, ma quello della variabile AK.
Esistono sul COMMODORE PLUS-4 tre tipi diversi di variabili con indice:

.MATRICI di variabili di tipo REALE (A)
.MATRICI di variabili di tipo INTERO (A%)
.MATRICI di variabili di tipo STRINGA (A$)
Una matrice puo' avere piu' di un indice, ad esempio B(3,2), che e' diverso da B(2,3), ha due indici. Non esistono limiti sul numero di indici che puo' avere una matrice, tranne il fatto che una matrice con molti indici occupa molta memoria.

Nel seguito useremo il nome MATRICE, che e' il piu' generale; di norma il nome VETTORE viene usato per matrici con un solo indice. A volte usiamo anche il termine inglese ARRAY.

Nel BASIC 3.5 del COMMODORE PLUS-4 gli indici iniziano dal valore 0, cioe' il primo elemento del gruppo corrisponde al valore 0 degli indici. L'uso di piu' indici consente di strutturare meglio il gruppo di variabili; per esempio una tabella di dati viene chiaramente definita con variabili a due indici, l'indice di riga, il primo, e l'indice di colonna, il secondo.

Quando si usano le matrici, bisogna definire alcuni parametri:
. il nome della matrice,
. il tipo di variabile,
. il numero di indici che individuano un elemento,
. il massimo valore che puo' raggiungere ogni indice, cioe' l'estensione del gruppo.
Tutte queste informazioni devono essere fornite al calcolatore prima di usare gli elementi della matrice, attraverso l'istruzione DIM, che e' l'istruzione di definizione delle matrici.
Quando noi eseguiamo:
DIM A(4,7,9)
informiamo il calcolatore che vogliamo definire una matrice di nome A, e che ha come elementi numeri REALI. Inoltre specifichiamo che la matrice ha tre indici (inseriamo infatti tre numeri nelle parentesi, separati da virgole). Infine specifichiamo che useremo:
. per il primo indice i valori da 0 a 4,
. per il secondo valori da 0 a 7,
. per il terzo valori da 0 a 9.
Se nel programma usi per gli indici valori superiori a quelli definiti, o un numero di indici diverso da quello specificato, ottieni il messaggio di errore: BAD SUBSCRIPT.
Per definire la matrice A di interi, con 100 elementi, scriveremo:
DIM A%(99)
Il vantaggio principale nell'usare matrici di tipo INTERO anziche' REALE e' che ogni elemento INTERO occupa 2 BYTE di memoria, mentre un elemento REALE occupa 5 BYTE.
Per definire il vettore QS di STRINGHE, con 65 elementi, scrive-

remo:
DIM QS$(64)
L'occupazione in memoria di un vettore di questo tipo non e' fissa: ogni elemento occupa 3 BYTE, piu' la lunghezza della stringa. Fai attenzione quando usi grosse matrici di stringhe, perche' potresti avere dei problemi di memoria.
Una volta dimensionata, una matrice non puo' piu' essere cancellata, se non con l'istruzione CLR, che cancella tutte le variabili: al massimo si puo' azzerarne il valore, ma non si puo' liberare completamente la memoria che occupa. Non e' neanche possibile cambiare le dimensioni di un vettore senza cancellare tutte le variabili (con CLR), altrimenti viene emesso il messaggio REDIM'D ARRAY (matrice ridimensionata).
Se si usa una matrice prima di averla dimensionata, come abbiamo fatto noi all'inizio di questo paragrafo, il calcolatore la dimensiona automaticamente a 10 per ogni dimensione. Un successivo dimensionamento produce ovviamente il messaggio REDIM'D ARRAY.

Le variabili con indice sono un elemento base della programmazione; esse consentono di utilizzare in pieno le possibilita' offerte dalle istruzioni per la gestione dei cicli

Gli indici possono essere numeri interi o reali, variabili numeriche intere o reali; di essi viene considerata solo la parte intera.

3.2 IL TRATTAMENTO DELLE STRINGHE

Un'interessante operazione che si puo' eseguire sulle variabili di tipo STRINGA e' quella del concatenamento: si puo' cioe' formare una stringa che contiene i caratteri di altre, due o piu', stringhe, poste una dopo l'altra. L'operatore che concatena le stringhe e' il "+".
Vediamo un esempio:
```
A$="CIAO":B$="COMMODORE":C$=A$+B$
PRINT C$
```
Il calcolatore stampa il contenuto della stringa C$, cioe' CIAOCOMMODORE.
Non esistono particolari precauzioni da usare nel concatenare le stringhe: bisogna pero' fare attenzione affinche' la nuova stringa non superi i 255 caratteri, caso in cui viene emesso il messaggio d'errore STRING TOO LONG.
Il concatenamento delle stringhe deve essere usato per avere in memoria una stringa piu' lunga di 88 caratteri; infatti con l'istruzione INPUT non si puo' leggere un dato stringa che superi 88 caratteri.

Trattiamo ora le FUNZIONI del BASIC che lavorano sulle stringhe, cioe' quelle funzioni che hanno come argomento una stringa o che danno come risultato una stringa.

Le variabili di tipo stringa contengono una sequenza di caratteri, che sono conservati nella memoria del calcolatore come numeri; ad ogni carattere corrisponde un codice numerico, il codice ASCII. Il codice del carattere A, ad esempio, e' 65. Il codice del carattere RETURN, e' 13. Tutti i caratteri che il calcolatore riconosce, anche quelli di controllo, hanno un codice. Nell'Appendice C sono descritti i codici di tutti i caratteri.

FUNZIONE ASC
La funzione fornisce un numero intero, che e' il valore del codice ASCII del primo carattere della stringa specificata; si scrive:
ASC(stringa), dove stringa puo' essere una costante tra virgolette o una variabile.
Se la stringa e' vuota la funzione ritorna 0.
Prova a digitare:
PRINT ASC("ABCD")
e il calcolatore stampa il numero 65, che e' il codice ASCII della lettera A, che e' il primo carattere che compone la stringa ABCD.

FUNZIONE CHR$
Questa funzione si puo' considerare l'inverso della funzione ASC. Infatti essa ritorna una stringa, il cui codice ASCII e' quello della variabile o costante numerica compresa tra parentesi. Ad esempio puoi digitare:
PRINT CHR$(65)
e il calcolatore stampa una "A".
Questa funzione e' molto importante, poiche' permette di stampare tutti i caratteri, anche quelli che non e' possibile comprendere tra virgolette, come ad esempio ESC e RETURN. L'argomento puo' variare tra 0 e 255 compresi.

FUNZIONE LEN
Questa funzione ha come argomento una stringa, costante o variabile, e fornisce in uscita un valore intero, che ne esprime la lunghezza. Ad esempio, la stringa "CIAO" e' lunga 4 caratteri, e l'istruzione:
PRINT LEN ("CIAO")
stampa il numero 4.
Anche gli spazi bianchi sono considerati ovviamente caratteri, come anche tutti i caratteri di controllo e di punteggiatura compresi nella stringa.

FUNZIONE LEFT$

Questa funzione ha due argomenti: la stringa su cui deve operare e un numero intero. Left significa sinistra, e LEFT$ fornisce una stringa, che e' la parte sinistra della stringa specificata, della lunghezza espressa dal numero specificato. Esempio:

```
PRINT LEFT$ ("CIAO COMMODORE",7)
```

stampa CIAO CO, cioe' i primi 7 caratteri a partire da sinistra della stringa "CIAO COMMODORE". Non esistono limitazioni sulla stringa: anche una stringa vuota e' accettata da questa funzione; l'unica limitazione sul numero intero e' che non deve essere negativo, e non deve superare 255.

FUNZIONE RIGHT$

Questa funzione e' sorella della precedente: right vuol dire destra, e RIGHT$ ritorna la parte destra della stringa specificata, della lunghezza espressa dal numero specificato. La sintassi e' la stessa di LEFT$. Esempio:

```
PRINT RIGHT$ ("CIAO COMMODORE",4)
```

stampa DORE, cioe' la parte destra della stringa "CIAO COMMODORE", della lunghezza di 4 caratteri. Le limitazioni sui parametri sono le stesse della funzione LEFT$.

FUNZIONE MID$

Questa funzione mette insieme le due precedenti, LEFT$ e RIGHT$. Essa lavora con 3 argomenti: una stringa e due numeri interi. La stringa e' quella su cui operare, e i due numeri indicano rispettivamente la posizione da cui partire (1=primo carattere da sinistra, 2=secondo, ecc.) e la lunghezza della nuova stringa. Esempio:

```
PRINT MID$("CIAO COMMODORE",9,4)
```

stampa MODO, che e' la stringa che parte dal nono carattere della stringa specificata, ed e' lunga 4 caratteri. La stringa, anche per MID$, puo' essere una qualunque stringa riconosciuta dal COMMODORE PLUS-4, il numero che indica la posizione di partenza deve essere compreso tra 1 e 255, il numero che esprime la lunghezza della stringa deve essere compreso tra 0 e 255.
Questa funzione puo' essere usata anche in altro modo, come pseudo variabile, e serve per modificare una parte di una stringa. Esempio:

```
A$="GATTO BRUTTO"
MID$(A$,7,6)="CARINO"
PRINT A$
```

stampa: GATTO CARINO
cioe' e' stato modificato il contenuto precedente di A$.

FUNZIONE INSTR

Questa funzione lavora su 2 parametri: due stringhe. La funzione cerca la seconda all'interno della prima stringa, se la trova,

ritorna un numero, che indica a partire da quale posizione la seconda stringa e' contenuta nella prima. Se la seconda stringa non e' contenuta nella prima, il numero e' zero. Esempio:
PRINT INSTR("COMMODORE PLUS-4","MODO")
stampa 4, poiche' la stringa "MODO" parte dal quarto carattere della stringa "COMMODORE PLUS-4".
A questa funzione si puo' anche fornire un altro parametro, un numero che esprime da quale carattere partire nella ricerca. Vediamo un esempio:
PRINT INSTR("COMMODORE PLUS-4","O")
stampa 2.
PRINT INSTR("COMMODORE PLUS-4","O",3)
stampa 5, infatti la ricerca e' iniziata dal terzo carattere, e il quinto carattere e' una O. Come hai notato, la ricerca viene fatta da sinistra verso destra.

FUNZIONE VAL
Questa funzione serve per passare da una variabile di tipo stringa, a contenuto numerico, a una di tipo numerico: e' chiaro che il calcolatore rifiuta di applicare funzioni matematiche a variabili di tipo STRINGA, anche se queste contengono numeri. La funzione VAL ritorna un numero, che esprime il valore numerico della stringa specificata. Esempio:
PRINT VAL("16")+1
stampa 17, che e' 16+1. Vi sono alcune regole, riguardo a questa funzione, che bisogna conoscere:
. Se la stringa inizia con un carattere non numerico, il suo valore e' zero:
A=5:PRINT VAL ("A")
stampa 0 anche se la variabile A vale 5.
. La funzione VAL non effettua operazioni matematiche sulla stringa:
PRINT VAL("1+1")
non stampa 2, ma 1. VAL infatti tiene conto solo dei caratteri numerici, cosi' come sono, fino al primo carattere non numerico.
Sono considerate stringhe numeriche quelle contenenti un qualunque numero con segno, anche espresso in formato esponenziale.
Questa funzione e' utile nelle operazioni di INPUT dove si vuole ricevere un numero, per evitare che il BASIC invii un messaggio d'errore quando viene introdotta una stringa anziche' un numero:
INPUT A$:A=VAL(A$)
e' meglio di:INPUT A; la conversione si ferma al primo carattere non numerico incontrato.

FUNZIONE STR$
Questa funzione si puo' considerare l'inverso della funzione VAL,

infatti riceve una variabile di tipo numerico, e ne ritorna una di tipo stringa. Puo' essere utile per applicare le funzioni LEFT$, MID$, RIGHT$, CHAR, alle variabili numeriche. Ad esempio:

```
CHAR 1,10,20,STR$(1234)
```

stampa la scritta 1234 in posizione 10,20, e non emette il messaggio TYPE MISMATCH, proprio perche' STR$ ritorna una variabile di tipo stringa.

3.3 LETTURA DATI DALL'INTERNO DEL PROGRAMMA

Il programma riceve i dati dall'esterno per mezzo dell'istruzione INPUT, oppure, carattere per carattere, con l'istruzione GET. A volte e' necessario incorporare in un programma blocchi di dati costanti; ogni singolo dato puo' essere assegnato a una variabile, ma devi scrivere molte istruzioni.
L'istruzione DATA si usa per inserire piu' comodamente dati nel programma. Per dati si intendono delle costanti, numeriche o stringhe, separate da virgole. Tutti i tipi di costanti usati dal COMMODORE PLUS-4 possono essere contenuti in linee DATA: numeri interi, reali in virgola fissa e in virgola mobile, stringhe. DATA e' un'istruzione che ha senso usare solo da programma, anche se in modo immediato non produce messaggio di errore.
I dati che sono archiviati in linee DATA vengono conservati nella memoria nell'ordine di comparizione delle relative linee DATA, che possono trovarsi ovunque in un programma. Per poter essere utilizzati essi devono essere trasferiti in variabili del programma, cioe' letti con l'istruzione READ. Esempio di alcune linee DATA:

```
1000 DATA 1,2,3,5,7,11,13,17,19:REM NUMERI PRIMI
1000 DATA A,E,I,O,U:REM VOCALI
1000 DATA "***   ***   ":REM SEGNI GRAFICI
1000 DATA1,A,2,B,3,C:REM UN NUMERO E UNA LETTERA
```

Anche per le linee DATA, come per la funzione VAL, non sono valide le espressioni matematiche. Le stringhe possono non essere comprese tra virgolette, a patto che non contengano caratteri di controllo o di interpunzione. Se invece sono comprese tra virgolette, possono contenere tutti i caratteri di controllo che e' consentito racchiudere tra virgolette. Una linea DATA, quando viene incontrata nel programma, viene saltata, come l'istruzione REM. A differenza di quest'ultima, pero', un successivo comando sulla stessa linea viene eseguito, anziche' essere ignorato. Esempio:

```
1000 DATA 1,2,3:PRINT "LINEA 1000"
```

stampa LINEA 1000 quando viene eseguito.
Naturalmente, a meno di non usare trucchi particolari, le linee

DATA devono essere introdotte in fase di stesura del programma, e non possono essere introdotte in fase di esecuzione. Per archiviare i dati che sono risultato di elaborazione non si puo' usare l'istruzione DATA, ma si deve ricorrere ai FILE (archivi) su nastro o su disco.

L'istruzione READ e' sorella di DATA, infatti e' quella che permette di leggere i dati che sono archiviati in linee DATA. I dati vengono letti in sequenza, uno dopo l'altro. La prima volta che viene incontrata un'istruzione READ, viene letto il primo dato presente nella prima linea DATA. La seconda volta, viene letto il secondo dato, e cosi' via. L'istruzione READ deve essere seguita dal nome di una o piu' variabili (che concordino con il tipo delle costanti da leggere): ogni variabile, dopo l'esecuzione dell'istruzione, conterra' il valore letto dalle linee DATA. Prova il seguente esempio:

```
1000 DATA 1,A,2,B,3,C,4,D
```

e poi, in modo diretto, esegui per 5 volte la seguente linea:

```
READA,A$:PRINTA;A$
```

Come vedi, la prima volta il calcolatore stampa 1 A, cioe' i primi due dati; la seconda volta stampa 2 B (la seconda coppia di dati), e cosi' via. La quinta volta stampa OUT OF DATA ERROR. I dati infatti sono terminati.

Il calcolatore pone al primo dato (contenuto nelle istruzioni DATA) un puntatore interno; esso avanza, dopo la lettura con READ, di un dato per ogni variabile letta.
L'istruzione RESTORE permette di riposizionare il puntatore o all'inizio del blocco di dati o a una determinata istruzione DATA del programma; cioe' consente di rileggere dati gia' letti o di saltarne un gruppo.
Anche CLR permette di rileggere i dati dall'inizio, ma cancella anche tutte le variabili. RESTORE, puo' essere seguito da un numero di linea, che indica la linea dove si vuole che il prossimo dato venga letto. Prova il programma:

```
1000 DATA 1,2,3,4,5
1010 DATA 6,7,8,9,0
```

e introduci in immediato la linea seguente:

```
RESTORE 1010:READA,B:PRINTA;B
```

il calcolatore stampa 6 7, perche' ha letto i dati a partire dalla linea 1010.

3.4 L'ISTRUZIONE GET

L'istruzione GET serve per leggere dati dalla tastiera carattere per carattere; essa ha una particolarita', legge comunque quando viene eseguita, quindi non segnala che vuole un carattere, e, se

non hai premuto alcun tasto, legge 0 se seguita da variabile numerica, stringa nulla, se seguita da variabile stringa.
Si scrive: GET variabile.
Essa puo' essere usata per leggere stringhe piu' lunghe di 88 caratteri (limite per INPUT), costruendo la stringa complessiva con l'operazione di concatenamento.

Un tipico uso di questa istruzione e' quello di creare un ciclo di attesa fino alla pressione di un particolare tasto:

```
500 GET A$:IF A$<>"P" THEN 500
```

prosegue solo se premi P.

Esiste un'altra istruzione la GETKEY, che fa proseguire solo se si preme un tasto; si scrive:

```
GETKEY lista variabili
```

per esempio:

```
600 GETKEY A$
```

non prosegue fino a quando non si preme un tasto. I tasti funzione danno errore.

3.5 LE FINESTRE SUL VIDEO

La finestra video e' una particolarita' che ti permette di definire una porzione di video dove lavorare: la zona al di fuori di questa porzione non viene interessata ne' da nuove scritte, ne' dall'operazione dello SCROLLING. Praticamente puoi decidere tu quante devono essere le righe e quante le colonne su cui scrivere, a patto di non superare le 25 righe e le 40 colonne, facendo in modo che tutto il resto dello schermo non sia alterato. Il programma ES3.1 ti mostra un'applicazione della finestra video: una parte dello schermo viene riservata a una scritta fissa, e il resto viene lasciato a tua disposizione per usare il calcolatore. Come puoi vedere, la scritta lampeggiante non va via, neppure se premi il tasto SHIFT-CLR/HOME.

```
1 REM ES3.1
10 PRINT CHR$(147);
20 PRINT"┌──────────────────────";
25 PRINT"──────────────────────┐"
30 PRINT"|              "CHR$(130)"COMMODORE PLUS-4
";
35 PRINTCHR$(132)"                 |"
40 PRINT"└──────────────────────";
```

```
45 PRINT"────────────────────┘"
50 CHAR,0,3,CHR$(27)+"T"
```

COME INSERIRE LA FINESTRA VIDEO
Abbiamo accennato nel Paragrafo 1.2 che la sequenza ESC T posiziona l'angolo in alto a sinistra della finestra video, mentre la sequenza ESC B ne posiziona l'angolo in basso a destra. Per posizionare l'angolo in alto a sinistra quindi e' sufficiente portare il cursore nel punto in cui lo si vuole posizionare, e premere i tasti ESC e poi T. In questo modo abbiamo selezionato uno dei due parametri che occorrono per definire la nostra finestra video: l'inizio. Per l'altro (la fine della finestra), ci posizioneremo col cursore dove abbiamo stabilito che si deve trovare l'angolo in basso a destra e premeremo ESC e poi B. Da questo momento non potremo piu' scrivere fuori dalla zona delimitata, se non eliminando l'effetto della finestra.
Per inserire la finestra da programma, bisogna portare il cursore nell'angolo della finestra, stampando i caratteri di controllo necessari, poi stampare CHR$(27) (che e' il codice ASCII di ESC) e "T" o "B" a seconda che si tratti dell'angolo iniziale, in alto a sinistra, o di quello finale, in basso a destra.

COME TOGLIERE LA FINESTRA VIDEO
Per eliminare la finestra, possiamo premere due volte di fila il tasto CLR/HOME. Ovviamente anche premendo il tasto RESET la finestra video si cancella, ma cosi' si cancella anche il programma contenuto in memoria; invece, RUN/STOP-RESET cancella la finestra video, senza cancellare il programma in memoria. Un altro modo per cancellare la finestra video e' premere i tasti ESC N; cosi' pero' tutto quello che si trova sullo schermo viene cancellato. ESC R produce automaticamente una finestra video di 23 righe X 38 colonne, e cancella tutto lo schermo.
Per togliere la finestra da programma, basta far esegire l'istruzione:
PRINT CHR$(19) CHR$(19)
che equivale a premere due volte di fila il tasto CLR/HOME.

3.6 DEFINIZIONE DELLE FUNZIONI UTENTE

ISTRUZIONE DEF FN
DEF FN significa DEFINE FUNCTION (definisci la funzione). Per funzione si intende una funzione matematica, cioe' una qualsiasi relazione che riceve in ingresso un valore e ne ritorna un altro. DEF FN va seguito dal nome della funzione e, tra parentesi, dal nome della variabile, su cui deve lavorare. La variabile su cui lavora la funzione si chiama ARGOMENTO. Ecco un esempio:
10 DEF FN F(X)=X*X

Abbiamo definito una funzione che eleva al quadrato l'argomento. Se infatti chiediamo il valore della funzione, aggiungendo la linea:

```
20 PRINT"-4 AL QUADRATO FA" FN F(-4)
```

e scriviamo RUN, vediamo che il calcolatore stampa:

```
-4 AL QUADRATO FA 16
```

Come hai notato, la X non e' una vera variabile; essa serve solo per definire la funzione. Quando la funzione viene richiamata, i calcoli sono eseguiti sull'argomento che viene passato in quel momento. Nel nostro esempio e' stata passata come argomento la costante -4.

La X contenuta tra parentesi nella definizione della funzione non ha relazione con una eventuale variabile X presente in altre parti del programma.

Nel COMMODORE PLUS-4 si puo' passare solo un argomento alla funzione: NON si puo' cioe' usare una funzione di piu' variabili del tipo:

```
DEF FN F(X,Y)=SQR(X*X+Y*Y)
```

ma bisogna passare una variabile solamente, tra le parentesi.

```
10 DEF FN F(X)=SQR(X*X+Y*Y)
```

In questo caso, potremo calcolare la funzione nel modo seguente:

```
20 Y=4: PRINT FN F(3)
```

In questo modo abbiamo calcolato il valore di una funzione di due variabili, passandone una tra parentesi, e l'altra attraverso la variabile Y. Il risultato di questo programma e' stampare 5, che e' l'ipotenusa di un triangolo rettangolo coi cateti che valgono 3 e 4.

Ricorda di porre la linea ove definisci la funzione all'inizio nel programma, perche' viene emesso il messaggio UNDEF'D FUNCTION (funzione non definita) se viene richiesto il valore di una funzione di cui non e' ancora stata incontrata la definizione:

```
10 PRINT FN Q(3)
20 DEF FN Q(Z)=Z
RUN
```

Il calcolatore si arresta col messaggio di errore UNDEF'D FUNCTION ERROR IN 10 perche' il programma ha trovato prima un'istruzione FN di una DEF FN. Scambiando il numero delle linee il programma funziona correttamente.

3.7 ALCUNE FUNZIONI AVANZATE

LA FUNZIONE RND(X)

Questa funzione ritorna un numero REALE pseudo CASUALE compreso tra 0 e 1 escluso. Per ottenere un numero compreso tra X e Y, bisogna moltiplicare il numero casuale per (Y-X) e sommare X. Se poi si vuole che il numero sia intero, si puo' usare la funzione INT. Scriviamo una linea che ritorni un numero intero casuale compreso tra 1 e 7, escluso 7:

```
PRINT INT(RND(1)*6+1)
```

Stampa un numero intero casuale compreso tra 1 e 7, escluso 7, cioe' tra 1 e 6 compresi.
Generare numeri casuali e' un problema abbastanza complicato, per una macchina precisa come un calcolatore. Occorre infatti avere una base casuale da cui partire. Nel COMMODORE PLUS-4 vi sono tre modi per ottenere le basi dei numeri casuali, che si selezionano scegliendo il valore dell'argomento della funzione:
.RND(0) produce il numero casuale partendo da un contatore interno al calcolatore, che viene continuamente e rapidamente incrementato: e' il modo nel quale il numero piu' si avvicina a un vero numero casuale
.RND(numero negativo) genera il numero casuale partendo dal numero tra parentesi:

PRINT RND(-1)
stampa sempre 2.99196472E-08
.RND(numero positivo) genera il nuovo numero partendo dal precedente numero casuale generato, permettendo di generare una serie di numeri casuale ripetibile:
PRINT RND(1), appena acceso il calcolatore, stampa 1.07870447E-03. La seconda volta stampa .793262171. Anche RND(4) stampa lo stesso risultato di RND(1), perche' il seme non dipende dal valore dell'argomento della funzione, se questo e' positivo.

LA FUNZIONE PEEK(X)
E' una funzione che dovrebbe essere usata da un programmatore che conosce a fondo la macchina. Ritorna il valore del byte di memoria di indirizzo X. X non puo' essere negativo, ne' maggiore di 65535, perche' il COMMODORE PLUS-4 puo' indirizzare 65536 byte diversi (tra 0 e 65535).

IL COMANDO POKE X,Y
E' il contrario di PEEK, cioe' anziche' leggere il contenuto del byte X, vi scrive il numero Y. Y deve essere compreso tra 0 e 25-5, e per il valore di X vedi PEEK.
POKE 16384,200:PRINT PEEK(16384)
stampa 200. Anche premendo NEW, CLR/HOME, il contenuto di questo byte non cambia. Poiche' la memoria viene usata dal calcolatore per molte funzioni, cambiare il contenuto di byte di memoria di cui non si conosce l'uso e' pericoloso, in quanto puo' facilmente causare l'arresto completo del calcolatore.

3.8 LA GESTIONE DEGLI ERRORI

Il COMMODORE PLUS-4 possiede diversi comandi che ti aiutano nel "debugging", cioe' la correzione, del programma.

Questi comandi sono HELP, TRON, TROFF, STOP. A volte ti puo' aiutare anche l'Appendice D, che riporta la maggior parte delle cause che procurano gli errori solitamente piu' "difficili" da scoprire.

IL COMANDO HELP
E' forse il comando piu' potente a disposizione per capire qual'e' l'istruzione che ha prodotto l'errore. Puoi ottenere di eseguire questo comando premendo il tasto di funzione con la scritta HELP, o digitando il comando da tastiera, dopo che si e' verificato un errore nel programma. Vedrai lampeggiare un pezzo di linea. La prima istruzione che lampeggia e' quella nella quale si e' verificato l'errore, quella cioe' che non e' stata portata a termine a causa dell'errore. HELP non funziona sui comandi immediati. Purtoppo capita anche ai programmatori piu' esperti di fare errori nei programmi. Le tecniche moderne di programmazione insegnano a scrivere programmi molto ordinati, in modo da dividere il programma in MODULI (parti il piu' possibile indipendenti). Tale tecnica permette di PROVARE separatamente ogni MODULO in tutte le condizioni piu' "critiche" per ciascun modulo.
Cerca di rendere i programmi ordinati e leggibili: risparmierai tempo in fase di correzione e di modifica.

IL COMANDO TRON
TRON vuol dire TRACE ON, cioe' "segna" i numeri di tutte le linee che esegui. In questo modo e' possibile avere una traccia di dove il programma e' passato, e capire qual'e' la causa di un malfunzionamento.

IL COMANDO TROFF
E' il contrario di TRON. Permette di evitare che il calcolatore continui a stampare i numeri di linea delle linee eseguite anche dopo che non occorre piu' stampare la traccia del programma.

IL COMANDO STOP
Questo comando permette di arrestare il programma in determinati punti "critici", per richiedere il valore di variabili importanti, o per vedere se il programma passa in certi punti. E' possibile riprendere l'esecuzione del programma dopo uno STOP, col comando CONT, a patto di non aver modificato il programma, di non avere commesso errori in modo immediato e di non aver eseguito istruzioni che distruggano il programma o le variabili (NEW, CLR, ecc.).

Il COMMODORE PLUS-4 ti da' la possibilita' di scrivere una routine di gestione degli errori, una routine cioe' che viene eseguita automaticamente quando viene incontrato un errore. E' ovvio che una routine che deve gestire gli errori e' un po'

"delicata", nel senso che puo' produrre un ciclo senza fine e inarrestabile, se gestita male: prova ad esempio il seguente programma":

```
10 TRAP 100
20 GOTO 20
100 RESUME
```

Puoi arrestare questo programma solo premendo il tasto di RESET.

IL COMANDO TRAP

TRAP e' il comando che permette di definire il numero di linea dove parte la routine di gestione degli errori. L'istruzione:

```
TRAP 20000
```

informa il calcolatore che in caso di errore deve andare alla linea 20000 anziche' emettere il messaggio di errore e arrestare l'esecuzione. Usato da solo, cioe' senza il numero di linea, TRAP toglie l'effetto "trappola" e ritorna nel modo normale, in cui in caso di errore viene emesso il messaggio di errore del sistema e viene arrestata l'esecuzione. In questo modo e' possibile inserire e disinserire l'effetto trappola lungo il programma, in modo da utilizzarlo solo in quei punti dove e' prevedibile un errore particolare. Tra tutti i messaggi di errore ce n'e' uno solo che non "sente" l'effetto trappola: UNDEF'D STATEMENT. Del resto in un programma corretto tale situazione non si dovrebbe mai verificare. TRAP inoltre si puo' usare solo in modo programma, e non in modo diretto.

LA ROUTINE DI GESTIONE DEGLI ERRORI

Abbiamo accennato che questa routine e' delicata: ricorda di gestire solo gli errori che hai previsto, e di non riprendere l'esecuzione del programma se l'errore non e' quello che avevi previsto. Vi sono nel COMMODORE PLUS-4 due variabili riservate, ER e EL, che indicano rispettivamente il numero di codice dell'errore e il numero della linea dove si e' verificato l'errore. Vi e' anche una funzione, ERR$(X), che ritorna una stringa contenente il messaggio di errore dell'errore che ha codice X. Scriviamo una ruotine degli errori che non ti permette di arrestare il programma col tasto RUN/STOP:

```
20000 IF ER <> 30 THEN PRINT ERR$(ER) "ERROR IN" EL: END
20010 PRINT "NON PUOI ARRESTARE IL PROGRAMMA SE NON CON RESET"
20020 RESUME
```

Una routine di errore cosi' non permette di arrestare il programma col tasto STOP (codice di errore = 30), ma stampa il messaggio di errore se si verifica qualunque altro errore. Il COMMODORE PLUS-4 e' costruito in modo da non gestire, se non in modo sistema, eventuali errori che si verificano nella routine di gestione degli errori: in tale circostanza viene emesso il messaggio di errore relativo all'ultimo errore verificatosi. Come probabilmente hai notato, tale routine assomiglia molto a una

subroutine, che quindi non deve terminare con GOTO. A diffetenza della subroutine, non deve terminare con RETURN, ma SEMPRE con RESUME.

IL COMANDO RESUME

Va sempre usato per terminare una routine di gestione degli errori, altrimenti il calcolatore ritiene di trovarsi ancora in tale routine, e non esegue piu' la routine degli errori quando se ne verifica un altro. Il calcolatore ENTRA in modo GESTIONE ERRORE quando si verifica un errore, e ne esce solo quando incontra RESUME. Vi sono tre diversi modi di usare RESUME:

.RESUME da solo riprende l'esecuzione del programma principale ripetendo l'istruzione nella quale si e' verificato l'errore

.RESUME NEXT riprende dall'istruzione successiva a quella dove si e' verificato l'errore

.RESUME numero linea riprende a partire dal numero di linea specificato.

CAPITOLO 4

COMUNICAZIONE CON LE PERIFERICHE

4.1 COME IL CALCOLATORE COMUNICA CON LE PERIFERICHE

Le periferiche sono dei dispositivi per l'ingresso e l'uscita dei dati, tramite i quali comunichiamo con il calcolatore. Ogni periferica utilizza un particolare tipo di supporto fisico per la comunicazione.
Alcuni di questi dispositivi di I/O (Input/Output) sono tali da consentire una comunicazione diretta con l'utente; nel caso del COMMODORE PLUS-4 abbiamo:
.la tastiera, che usa il supporto tasti,
.il televisore (o il monitor), che usa il supporto schermo video,
.la stampante, che usa il supporto carta,
.i joystick, che usano come supporto leve o bottoni o rotori.
Altri non consentono una comunicazione diretta, in quanto utilizzano supporti fisici di tipo magnetico; per il COMMODORE PLUS-4 abbiamo:
.il registratore a cassetta,
.l'unita' a floppy disk.

I dispositivi di I/O sono apparecchiature distinte dal calcolatore, anche se esso, come il COMMODORE PLUS-4, si trova inscatolato in una di esse, la tastiera. Per consentire la comunicazione tra il calcolatore e ogni dispositivo di I/O e' necessaria un'interfaccia di collegamento. Le interfacce di I/O sono, in generale, dei dispositivi programmabili, cioe' contengono al loro interno dei registri, ai quali si puo' accedere mediante indirizzi, come se fossero byte di memoria, programmando il funzionamento dell'interfaccia stessa. Dal punto di vista fisico le interfacce di I/O possono gia' essere contenute all'interno del calcolatore o delle periferiche. Per il COMMODORE PLUS-4 le interfacce per le periferiche standard COMMODORE sono gia' contenute nel calcolatore.

Ogni periferica di I/O e' individuata mediante un numero intero che, nell'Appendice A, abbiamo chiamato "unita'". Questo numero nella letteratura viene spesso chiamato "dn", da Device Number (numero apparecchiatura). Per alcune periferiche il numero "dn"

puo' essere modificato agendo su alcuni switch, interni o esterni; per altre esso e' fisso; i valori di "dn" validi sono riportati nel Paragrafo A.4.
Per i joystick non esiste un "dn", la comunicazione risulta sempre aperta.

Per le periferiche collegate tramite l'interfaccia seriale, stampante e disco, deve essere definito un numero che identifica il modo della comunicazione. Questo numero e' stato chiamato "sa" nella descrizione del BASIC; esso puo' variare da 0 a 15. Il numero "sa" ha diverso significato a seconda delle periferiche; nei Capitoli 5 e 6 trattiamo diffusamente stampante e disco.

L'insieme dei dati che vengono scambiati con una periferica si chiama "stream" o "flusso". Esso si trasforma in Output in una registrazione sul supporto fisico che viene chiamata "file". In Input il file, gia' registrato sul supporto fisico, alimenta il flusso di dati verso il calcolatore. Ogni file e' contraddistinto da un numero, chiamato "lfn", da Logical File Number (numero logico file). Il "lfn" puo' variare da 0 a 255; per alcune periferiche una parte dell'intervallo di variabilita' del "lfn", o alcuni valori, possono avere un significato particolare.
Molte istruzioni BASIC usano il "lfn", che viene definito con l'istruzione OPEN.
Un file e' composto da RECORD, cioe' da singoli gruppi di registrazioni che ha senso considerare insieme. Si chiamano CAMPI le singole registrazioni che compongono un record. Dobbiamo fare una distinzione tra:
.record logici,
.record fisici.
Le dimensioni dei record, logici e fisici, e dei campi si misurano in numero di caratteri o in byte.
Il RECORD LOGICO e' composto da tutti i campi che interessano l'argomento a cui il record si riferisce; in conseguenza puo' essere lungo quanto e' necessario (almeno in linea teorica).
Il RECORD FISICO dipende dal tipo di supporto di registrazione usato; per esempio su un tipo di carta e con una stampante di un certo tipo non si possono scrivere piu' di 80 caratteri per riga. In questo caso il record fisico, riga di stampa, e' di 80 caratteri. Possiamo usare quella stampante con quella carta per scrivere i dati di un record logico di 300 caratteri; il record logico usera' piu' record fisici. Analogamente una linea del video contiene 40 caratteri, ma noi possiamo scrivere sul video un record logico lungo che sta su piu' righe.
In generale esiste un limite nelle dimensioni dei record fisici, ed esso dipende anche dal supporto di registrazione.
Altre caratteristiche che vengono in generale precisate per i fi-

le sono le seguenti:
.record logici di lunghezza fissa o variabile,
.numero dei campi, componenti il record, fisso o variabile,
.lunghezza dei singoli campi fissa o variabile,
.tipo del file: sequenziale, diretto, relativo, con indice sequenziale,
.metodo di accesso ai record del file.

Un file e' di tipo SEQUENZIALE quando viene creato scrivendo i record uno dopo l'altro, partendo dal primo. Esso e' di tipo DIRETTO quando si puo' scrivere un record logico conoscendo la sua posizione fisica sul supporto di registrazione, indipendentemente dalla scrittura del record precedente. Un file RELATIVO viene scritto record per record (anche non in sequenza) fornendo il numero d'ordine del record nel file (primo, terzo,.-..). Un file CON INDICE SEQUENZIALE viene scritto un record dopo l'altro, creando contemporaneamente un indice sequenziale (file separato dal file principale) che consente l'accesso diretto mediante una chiave al record del file principale; la chiave e' il valore di un determinato campo.
Il metodo di accesso ai record di un file dipende dal tipo di file, dal supporto di registrazione usato e dal software disponibile.
La gestione software delle periferiche puo' avvenire solo per mezzo delle apposite routine del SISTEMA OPERATIVO del calcolatore, oppure puo' dipendere anche da routine memorizzate nelle ROM delle periferiche. Nel nostro caso la stampante e l'unita' disco contengono del software registrato in ROM.

Riepiloghiamo le istruzioni del BASIC che riguardano la gestione delle periferiche come file:

OPEN lfn[,dn[,sa[,"nomef,tipo,modo"]]]
per stabilire la comunicazione

CLOSE lfn
per chiudere la comunicazione

CMD lfn[,lista variabili]
per trasferire a una periferica diversa l'uscita video

GET# lfn,lista variabili
per leggere dati carattere per carattere

INPUT# lfn,lista variabili
per leggere dati variabile per variabile

PRINT# lfn,lista variabili
per scrivere dati

Il sistema raccoglie in una tabella le informazioni relative ai file aperti; tale tabella puo' contenere 10 elementi.

TABELLA GESTIONE FILE

Num.elem.	lfn	dn	sa
. 1)	1289	1299	1309
. 2)	1290	1300	1310
. 3)	1291	1301	1311
. 4)	1292	1302	1312
. 5)	1293	1303	1313
. 6)	1294	1304	1314
. 7)	1295	1305	1315
. 8)	1296	1306	1316
. 9)	1297	1307	1317
.10)	1298	1308	1318

La tabella dei file aperti viene utilizzata in modo che le informazioni sui file aperti occupino le prime posizioni. Quando un file viene chiuso, se e' l'ultimo nella tabella, essa resta come e' e viene aggiornato solo il puntatore (byte 151) all'ultima posizione occupata, se non e' l'ultimo, la sua posizione viene occupata dall'ultimo file e viene aggiornato il puntatore.

Il programma ES4.1 che segue mostra i contenuti di questa tabella in diverse situazioni, e i contenuti dei byte che danno informazioni circa il file corrente e il numero dei file aperti. Questi ultimi sono:

byte 171: lunghezza nome file corrente
byte 172: lfn file corrente
byte 173: sa file corrente
byte 174: dn file corrente
byte 175/176: puntatore al nome del file
byte 151: numero file aperti e contemporaneamente puntatore all'ultima posizione occupata nella tabella.

```
1 REM ES4.1
2 OPEN4,4:CMD4
3 GOSUB100
10 OPEN0,0
15 OPEN1,3
20 OPEN2,0
21 PRINT#4:CLOSE4
23 OPEN5,1,1,"PIPPO"
24 OPEN4,4:CMD4
25 GOSUB100
30 PRINT#4:CLOSE4
```

```
40 CLOSE0:CLOSE1:CLOSE2:CLOSE5
43 OPEN4,4:CMD4
45 GOSUB100
47 PRINT#4:CLOSE4
50 STOP
100 REM STATO FILE
101 PRINT"NUMERO FILE APERTI: ";PEEK(151)
103 PRINT"FILE CORRENTE"
104 PRINT"LFN","DN","SA"
105 PRINTPEEK(172),PEEK(174),PEEK(173)
110 PRINT"PUNTATORE NOME: ";256*PEEK(176)+PEEK(1
75)
113 PRINT"LUNGHEZZA NOME: ";PEEK(171)
115 PRINT:PRINT"TABELLA FILE"
117 PRINT"LFN","DN","SA"
120 FORK=1289TO1298
125 PRINTPEEK(K),PEEK(K+10),PEEK(K+20)
130 NEXTK
135 RETURN
```

COMMENTO A ES4.1

.2/3: viene aperta la stampante e stampata la situazione.

.10/25: vengono aperti i file con lfn 0, 1 e 2, poi viene chiusa la stampante e aperto un file su cassetta con lfn=5. Viene riaperta la stampante e stampata la situazione.

.30/45: vengono chiusi tutti i file aperti, poi riaperta la stampante e stampata la situazione.

.47/50: viene chiusa la stampante e il programma termina.

.100/135: sottoprogramma che stampa il valore dei puntatori e la tabella.

Osservando i risultati puoi verificare come viene gestita la tabella.

RISULTATI ES4.1

```
NUMERO FILE APERTI:  1
FILE CORRENTE
LFN            DN             SA
 4             4                255
PUNTATORE NOME:  250
LUNGHEZZA NOME:  0

TABELLA FILE
LFN            DN             SA
 4             4                255
 0             0                0
 0             0                0
 0             0                0
 0             0                0
```

```
 0              0              0
 0              0              0
 0              0              0
 0              0              0
 0              0              0



NUMERO FILE APERTI:  5
FILE CORRENTE
LFN            DN            SA
 4              4              255
PUNTATORE NOME:  250
LUNGHEZZA NOME:  0

TABELLA FILE
LFN            DN            SA
 2              0              96
 0              0              96
 1              3              255
 5              1              97
 4              4              255
 0              0              0
 0              0              0
 0              0              0
 0              0              0
 0              0              0



NUMERO FILE APERTI:  1
FILE CORRENTE
LFN            DN            SA
 4              4              255
PUNTATORE NOME:  250
LUNGHEZZA NOME:  0

TABELLA FILE
LFN            DN            SA
 4              4              255
 5              1              97
 1              3              255
 5              1              97
 4              4              255
 0              0              0
 0              0              0
 0              0              0
 0              0              0
 0              0              0
```

4.2 LA TASTIERA VISTA COME FILE

Il BASIC fornisce le istruzioni per la tastiera, che e' considerata la periferica di Input principale. Il byte 152 contiene il numero di default della periferica di Input; esso all'accensione e' zero, "dn" della tastiera. I comandi INPUT, GET e GETKEY attendono Input dalla tastiera. Essa e' una periferica un po' particolare, infatti pur essendo di Input, le routine del SISTEMA OPERATIVO relative al suo uso forniscono per alcune istruzioni (INPUT) anche un Output sul video, che, ovviamente, potrebbe essere evitato, usando routine diverse.
Vediamo ora, con alcuni esempi, come si puo' usare la tastiera come una periferica gestita come file.

Nel programma ES4.2 mostriamo un primo esempio.

```
1 REM ES4.2
10 OPEN 1,0
15 PRINT"SCRIVI NUMERO: ";:INPUT#1,N
20 PRINT "▮▮- HAI SCRITTO: ";N
25 CLOSE1
```

COMMENTO A ES4.2
.10: apriamo la dn=0 come file con lfn=1, trascurando gli altri parametri della OPEN, che non risultano necessari.
.15: con la PRINT scriviamo il messaggio di richiesta di un numero e terminiamo con ";" per non andare a capo. Con la INPUT#1, leggiamo dal file logico 1 un numero, cioe' dalla tastiera. Quando esegui il programma non vedi il "?" di richiesta dati. Termini il numero come al solito con RETURN, vedi scomparire il cursore, ma esso non va a nuova linea.
.20: stampiamo un messaggio e il numero N. Il messaggio tra virgolette inizia con due caratteri "cursore a destra" perche' altrimenti esso verrebbe scritto sopra l'ultima cifra del numero; ne abbiamo messi due per creare anche uno spazio. Possiamo concludere che il RETURN di chiusura dato lascia in questo caso il cursore sull'ultimo carattere scritto.
Se rispondi con un numero di parecchie cifre, il messaggio successivo all'INPUT andra' in parte a nuova linea.

Nel programma ES4.3 riportiamo un altro esempio.

```
1 REM ES4.3
10 OPEN 170,0
15 PRINT"SCRIVI NUMERO: ";:INPUT#170,N
20 PRINT:PRINT "HAI SCRITTO: ";N
25 CLOSE170
```

La differenza di ES4.3 rispetto al programma precedente e' che abbiamo usato lfn=170 e che alla linea 20, andiamo a capo con la

prima PRINT e, poi, scriviamo il messaggio; abbiamo potuto eliminare i caratteri di "cursore a destra".

Nel programma ES4.4 riportiamo un terzo esempio.

```
1 REM ES4.4
10 OPEN 0,0
15 PRINT"SCRIVI NUMERO: ";:INPUT#0,N
20 PRINT "HAI SCRITTO: ";N
25 CLOSE0
```

In questo caso abbiamo usato lfn=0, questo produce il solito effetto di INPUT dalla tastiera, cioe' compare il punto interrogativo di richiesta dati e il RETURN di chiusura dato manda a nuova linea. L'istruzione INPUT normale corrisponde ad aver usato una OPEN 0,0 seguita da una INPUT#0.

4.3 IL VIDEO VISTO COME FILE

L'istruzione PRINT del BASIC manda i dati sul video, che e' considerato la periferica di Output principale, a partire dalla posizione del cursore; la posizione di stampa puo' essere modificata usando i caratteri di controllo del cursore e le funzioni TAB e SPC. Il byte 153 contiene il numero di default della periferica di Output; esso all'accensione e' 3, "dn" del video. Vediamo ora, con alcuni esempi, come si puo' usare il video, gestendolo come un file, sia in Input che in Output.

Nel programma ES4.5 mostriamo un primo esempio.

```
1 REM ES4.5
10 OPEN3,3
15 PRINT#3,"♥";"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
16 PRINT#3,1234
17 A$=""
20 PRINT#3,"S";
23 FORK=1TO26:GET#3,B$
25 A$=A$+B$
30 NEXTK:PRINT#3,"SQ";
31 N$="":GET#3,N1$
33 GET#3,N1$:IFN1$=CHR$(32)THEN35
34 N$=N$+N1$:GOTO33
35 PRINT#3,"SQQQQQQQQQQQQQ"A$
37 PRINT#3,VAL(N$)
40 CLOSE3:STOP
```

COMMENTO A ES4.5

.10: apre il video, dn=3, con lfn=3.

.15: stampa sulla prima linea del video pulito, dopo aver usato il carattere di controllo SHIFT-CLEAR/HOME, le 26 lettere dell'alfabeto, con PRINT#3.

.16: stampa con PRINT#3 sulla seconda linea del video il numero 1234. Nota che dopo la stampa il cursore va a capo, ma sul video non viene registrato il carattere CHR$(13). Il video e' stato usato come file di Output.

.17: pone A$ a stringa nulla.

.20: stampa con PRINT#3 il carattere di controllo CLR/HOME per portare il cursore nella prima posizione del video.

.23/30: legge con GET#3, quindi usa il video come file di Input, quello che sta sulla prima linea, carattere per carattere e somma i caratteri letti in A$; questo per 26 volte. Poi con PRINT#3 stampa i caratteri di controllo CLR/HOME e FRECCIA GIU', per posizionarsi all'inizio della seconda riga del video.

.31: pone N$ a stringa nulla e legge il primo carattere, che sappiamo essere uno spazio, a vuoto. Se vogliamo leggere un numero con il segno meno, questo primo carattere va posto nella stringa N$.

.33: legge in N1$ un carattere; se esso e' uno spazio, cioe' il numero e' terminato, prosegue dalla linea 35.

.34: somma il carattere letto nella stringa N$ e torna alla linea 33.

.35: si posiziona sulla 13-esima riga del video e stampa A$ con PRINT#3.

.37: stampa con PRINT#3, sulla linea seguente il valore del numero letto.

.40: chiude il file con lfn=3 e si ferma.

Come vedi in questo esempio il video e' stato aperto come file ed usato alternativamente in Input e in Output. Non si puo', pero', usare l'istruzione INPUT#3, infatti da' l'errore "string too long", dato che non trova il carattere CHR$(13).

4.4 I FILE SU CASSETTA

Nel caso della cassetta e' possibile registrare facendo scorrere il nastro in sequenza, si tratta di registrazione sequenziale. Ovviamente se una registrazione e' di tipo sequenziale, anche la conseguente rilettura puo' essere solo di tipo sequenziale. Possiamo schematizzare il nostro file, in un pezzo di nastro magnetico, sul quale si susseguono i record, ognuno diviso in campi, come in Figura 4.1.

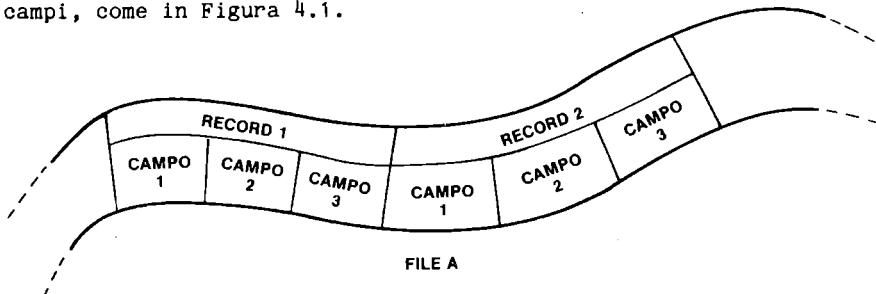



Figura 4.1 File, record e campi

Un file puo' essere o LETTO o SCRITTO, non si possono mescolare le due operazioni.
Non esistono limiti alla lunghezza dei singoli campi (salvo le limitazioni gia' note sulla grandezza dei numeri e la lunghezza delle stringhe).
Risulta abbastanza laboriosa l'operazione di aggiornamento di un file; infatti non si puo' correggere una parte di registrazioni. Per aggiornare un file su cassetta e' necessario trasferire il suo contenuto in memoria, e poi scrivere un nuovo file, modificando dove necessario. In conseguenza ti raccomandiamo di non creare file di dati troppo lunghi; devi sempre fare i conti con la memoria disponibile nel calcolatore.
Se vuoi organizzare un archivio di indirizzi, per esempio, invece di creare un unico file di nominativi, puoi spezzarlo in una serie di file, ognuno dedicato ai nomi che iniziano con un gruppo di lettere.

Le operazioni per la gestione dei file sono:
.1) apertura della comunicazione con il file,
.2) scrittura o lettura del file,
.3) chiusura della comunicazione con il file.

Al momento della creazione di un file, gli assegnamo un nome; questo nome va ricordato per poter comunicare con il file e viene registrato all'inizio del file. Il file e' visto dal programma anche con un numero distintivo, il "lfn", da Logical File Number; esso puo' variare da 1 a 255 (di norma si usano numeri piccoli).
Il parametro "sa" da Secondary Address, serve per precisare il tipo dell'operazione. Nel seguito troveremo:
.fn, nome file, stringa da 1 a 16 caratteri,
.lfn, numero logico del file, da 1 a 255,
.dn, numero della periferica, 1 per la cassetta,
.sa, indirizzo secondario, che per la cassetta puo' valere:
..0 o mancare per operazioni di lettura da nastro,
..1 per operazioni di scrittura su nastro, con registrazione del carattere di fine file,
..2 per operazioni di scrittura su nastro con registrazione anche di un altro carattere particolare alla fine del file, per segnalare anche la fine del nastro.
I parametri dn, lfn e sa, possono essere costanti o variabili numeriche; fn puo' essere una costante o una variabile stringa.
Per stabilire la comunicazione con un file si usa l'istruzione OPEN; essa si scrive:

```
OPEN lfn,dn,sa,fn
```

dove i parametri hanno il significato visto sopra; dopo la OPEN si puo' leggere o scrivere. Il tipo di operazione consentito sul

file dopo l'apertura dipende dal valore di "sa".
Se il file e' stato aperto per lettura, e quindi deve esistere gia' sul nastro, quando tenti di scriverci sopra ottieni un messaggio di errore.
In conseguenza del valore di "sa" al momento dell'esecuzione della OPEN compare sul video il messaggio di richiesta:
PRESS PLAY ON TAPE, oppure
PRESS PLAY & RECORD ON TAPE.
Quando viene eseguita la OPEN, dopo la risposta al messaggio di sistema, il video si sbianca, il nastro gira, e, in caso di lettura viene ricercata l'intestazione del file richiesto, mentre in caso di scrittura viene scritta la testata del nuovo file.

Le altre istruzioni per la gestione del file si riferiscono al file aperto per mezzo del numero "lfn".

Per scrivere si usa l'istruzione PRINT#; essa si scrive:

PRINT#lfn, lista dati

Scrive sul file aperto con numero logico "lfn" i dati della lista, che possono essere variabili o costanti. In particolare lista dati puo' essere una sola variabile.
E' importante sottolineare che i dati vengono registrati carattere per carattere sul nastro, e che e' necessario registrare un "carattere separatore" che, in fase di lettura, segnali la fine di un dato. Tale carattere separatore deve essere o la virgola o il RETURN. Il carattere separatore virgola puo' essere passato come: "," oppure CHR$(44), mentre il RETURN puo' essere passato come CHR$(13), oppure dall'assenza di punteggiatura dopo l'ultimo dato della lista. Esempi:

```
10 PRINT#3,"alba";"serena"
```

da' luogo in lettura a un solo dato che appare come "albaserena", infatti il separatore punto e virgola non ha fatto aggiungere spazi e non e' presente un separatore registrabile, mentre la mancanza di punteggiatura finale ha fatto registrare un RETURN.

```
10 PRINT#3,"alba","serena"
```

da' luogo in lettura a un solo dato che appare come "alba   serena", infatti il separatore virgola ha fatto aggiungere 8 spazi.

```
10 PRINT#3,"alba";",";"serena"
```

da' luogo in lettura a due dati: il primo "alba" e il secondo

"serena", infatti e' stata registrata la virgola separatrice. Tali dati pero' possono essere letti tutti e due solo da un'istruzione di lettura che contenga due variabili nella lista. Infatti una operazione di lettura legge sempre tutti i dati compresi tra due caratteri RETURN; se non poni un numero sufficiente di variabili nella lista di lettura puoi perdere dei dati. Questo non succede se il separatore e' il RETURN per ogni singolo dato.

```
10 PRINT#3,"alba":PRINT#3,"serena"
```

da' luogo in lettura a due dati, che sono stati registrati con il carattere separatore RETURN, infatti manca la punteggiatura alla fine della lista dati.

Durante le operazioni di scrittura di un file non avviene un reale trasferimento di dati tra calcolatore e cassetta ogni volta che si esegue una PRINT#; l'operazione di trasferimento dati avviene solo quando, dopo l'esecuzione di un certo numero di istruzioni PRINT#, e' stata riempita la zona di memoria che serve per immagazzinare i dati per la cassetta; tale zona si chiama BUFFER. Il buffer usato dal sistema si trova dal byte 819 al byte 1010 ($0333/$03F2); sono 192 byte. La gestione del buffer risulta trasparente per l'utente, che tratta i dati solo a livello logico. Se desideri vedere come i dati sono registrati nel buffer, per esempio dopo la OPEN di un file, o in qualunque altro momento, puoi leggere il contenuto del buffer usando la funzione PEEK.
Inoltre se vuoi vedere i dati presenti nel buffer carattere per carattere li puoi leggere con l'istruzione GET#.
Anche i file di programma transitano dal buffer sia in fase di lettura (LOAD), che in fase di scrittura (SAVE). Risulta meno agevole analizzare il contenuto del buffer per queste operazioni, dato che esse sono svolte con continuita' e, alla fine, nel buffer si trova solo l'ultima parte dei dati transitati.
In fase di scrittura quando il buffer e' pieno, il suo contenuto viene trasferito sulla cassetta; durante il trasferimento il video si sbianca.

Per leggere un file si usa l'istruzione INPUT#; essa si scrive:

```
INPUT#lfn,lista dati
```

e lista dati deve contenere i nomi delle variabili nelle quali leggere da nastro, separate da virgole. La INPUT# legge i dati che sono registrati tra due RETURN, perdendone alcuni, se il numero di variabili della lista non e' sufficiente. Inoltre il numero dei caratteri tra due RETURN non deve superare 88.
Il tipo delle variabili deve concordare con il tipo dei dati da leggere; in caso contrario si ha segnalazione di errore.

Per poter leggere con maggior liberta', si puo' usare l'istruzione GET#, che legge carattere per carattere, compresi i caratteri separatori registrati; essa si scrive:

GET#lfn,lista dati

E' consigliabile usare nomi di variabili stringa per evitare segnalazioni di errore.
Con GET# si possono leggere campi lunghi a piacere, dato che si leggono un carattere per volta.

Anche per le operazioni di lettura il trasferimento dei dati avviene a blocchi; alla prima lettura viene riempito il buffer e vengono poi prelevati da esso i caratteri. Il buffer viene riempito nuovamente quando tutti i dati presenti sono stati letti.

Per chiudere la comunicazione con un file si usa l'istruzione CLOSE; essa si scrive:

CLOSE lfn

ed e' necessario usarla per lavorare correttamente. Chiudere un file, aperto per scrivere, e' necessario anche per svuotare il buffer di eventuali caratteri residui. L'operazione di chiusura serve anche per liberare una posizione nella tabella di gestione dei file del BASIC; tale tabella consente di gestire solo 10 file contemporaneamente.

Riportiamo come esempio di scrittura di un file il programma CREAFILE.

```
1 REM CREAFILE
10 OPEN 3,1,1,"NOMI"
15 FORK=1TO10
20 INPUT"NOME=";N$
25 PRINT#3,N$
30 NEXTK
40 CLOSE 3
```

In esso alla linea 10 viene aperto il file NOMI per scrivere e gli viene assegnato lfn=3. Dalla linea 15 alla 30 si ha un ciclo nel quale vengono chiesti 10 nomi dalla tastiera e scritti sul nastro. Alla linea 40 viene chiuso il file.

Segue il programma esempio LEGGIFILE, nel quale viene aperto in lettura il file NOMI, vengono letti in ciclo e stampati sul video i 10 nomi e poi viene chiuso il file.

```
1 REM LEGGIFILE
10 OPEN 3,1,0,"NOMI"
15 FORK=1TO10
20 INPUT#3,N$
25 PRINT N$
30 NEXTK
40 CLOSE 3
```

In quest'ultimo esempio abbiamo letto 10 dati, infatti sapevamo di averne scritti 10. Se avessimo cercato di leggerne piu' di 10 avremmo avuto segnalazione di errore. Quando il file, aperto in scrittura, viene chiuso, il sistema aggiunge in fondo un carattere particolare che segnala la fine del file e si chiama EOF (da End Of File); anche il carattere EOT (da End Of Tape) se sa=2. Il carattere EOF, che e' uno 0 (codice ASCII 0) viene sentito quando si legge l'ultimo dato che lo precede, e viene registrato automaticamente dal sistema nella variabile riservata ST il numero 64. Per questa ragione dopo ogni lettura e' bene memorizzare ST in una variabile di comodo, da analizzare prima di tornare a leggere. Il carattere EOF viene prodotto da te se scrivi su un file il carattere CHR$(0); quindi devi evitare di usarlo, dato che il calcolatore lo riconosce come segnalazione di fine file. Il programma LEGGIFILE, e' meglio sia scritto come LEGGICONST, che segue.

```
1 REM LEGGICONST
10 OPEN#3,1,0,"NOMI"
15 INPUT#3,N$
20 TS=ST
25 PRINTN$
30 IFTS=0THEN15
40 CLOSE3
```

In esso, dopo l'istruzione INPUT#3, memorizziamo in TS la variabile di stato ST. Alla linea 30 analizziamo TS e torniamo a leggere dal file solo se essa e' a zero, altrimenti chiudiamo il file.

Nel caso dei file su cassetta non sono applicabili in modo rigoroso i concetti di RECORD e di CAMPO. Infatti potresti considerare RECORD la parte di registrazioni comprese tra due RETURN, e CAMPI i dati, separati da virgola, all'interno del record, ma avresti le seguenti limitazioni:

. il RECORD non potrebbe contenere piu' di 88 caratteri,
. il RECORD dovrebbe essere sempre letto da una sola INPUT#, usando un numero di variabili pari al numero dei CAMPI, oppure dovresti usare GET# e leggere carattere per carattere.
Per tutte queste ragioni, in generale i file sequenziali su cassetta si registrano usando il RETURN (CHR$(13)) sia come separatore di campo che come separatore di record.

ESEMPIO ARCHIVIO DI DATI SU CASSETTA

Riportiamo, come esempio riassuntivo, un programma che consente di gestire un archivio di dati su cassetta. Costruiamo il programma in modo che sia facilmente modificabile, e tu lo possa adattare alle tue esigenze, cambiando sia i nomi dei campi, che il loro numero.

Le operazioni che deve essere possibile effettuare per gestire un archivio di dati sono le seguenti:

.a) creazione ex novo dell'archivio,
.b) aggiornamento dell'archivio:
   b1) cancellazioni di record,
   b2) aggiunta di nuovi record,
   b3) modifica di record gia' esistenti,
.c) lista di tutto o parte dell'archivio.

Queste operazioni sono in generale necessarie per qualunque archivio; inoltre ogni archivio puo' essere utilizzato per altri scopi, che dipendono dalle specifiche esigenze inerenti ai soggetti archiviati.

Nella Figura 4.2 riportiamo uno schema a blocchi molto generale del programma FILESEQ, per la gestione di un archivio sequenziale.

Noi abbiamo deciso di creare un archivio formato da record di 6 campi ciascuno, relativi a un'agenda di indirizzi. Creiamo e manteniamo l'archivio in ordine alfabetico in base ai due primi campi; in conseguenza i dati, nella fase di creazione iniziale del file devono essere forniti rispettando tale ordine. Il programma rifiuta nominativi fuori ordine, come pure nominativi uguali.
All'inizio viene proposto un menu' per scegliere tra le tre funzioni principali: CREAZIONE, AGGIORNAMENTO, STAMPA. Viene chiesto quanti record si vogliono trattare, per predisporre il dimensionamento delle variabili.
La funzione di AGGIORNAMENTO inizia trasferendo completamente in

Figura 4.2 Diagramma a blocchi FILESEQ

memoria il file esistente; poi si articola in tre fasi:
.1) modifica in memoria dei record gia' esistenti,
.2) cancellazione in memoria, ponendo spazi al posto del primo campo, nei record da eliminare dal file,
.3) aggiunta, durante la riscrittura del file, di eventuali nuovi record, che devono essere forniti in ordine.

In Figura 4.3 riportiamo uno schema a blocchi della fase di aggiornamento.

```
1 REM FILESEQ
3 REM CREAZIONE E GESTIONE FILE SEQUENZIALE
5 REM RECORD IN ORDINE PRIMI DUE CAMPI
7 PR=3:REM DN PERIFERICA DI STAMPA
9 NF=4:REM NUMERO LOGICO FILE DI STAMPA
11 NC=6:REM NUMERO CAMPI DEL RECORD
13 REM VETTORI DESCRIZIONI E DATI
15 DIMD$(NC),I$(NC)
17 CH$=CHR$(13):SP$="   "
19 REM DESCRIZIONI CAMPI
```

```
21 DATA COGNOME, NOME,INDIRIZZO
23 DATA CAP,CITTA',TEL.
25 FORK=1TONC:READD$(K):NEXTK
27 REM SCELTA OPERAZIONI
29 PRINT"                SCELTA OPERAZIONE"
31 PRINTTAB(10)"1) CREAZIONE"
33 PRINTTAB(10)"2) AGGIORNAMENTO"
35 PRINTTAB(10)"3) STAMPA"
37 PRINTTAB(10)"9) FINE"
39 GETKEYR$:R=VAL(R$)
41 IFR<1OR(R>3ANDR<>9)THEN39
43 IFR=9THENGOSUB365:STOP
45 PRINT"      QUANTI RECORD TRATTI IN TUTTO,
"
47 PRINT"     SE IL FILE NON ESISTE 0,"
49 PRINT"     SE IL FILE ESISTE GIA'?"
51 INPUT"NUMERO RECORD: ";N
53 INPUT"CONFERMI (S/N): ";R$
55 IFR$<>"S"THEN45
57 REM DIMENSIONAMENTO MEMORIA PER N RECORD
59 DIMMC$(N,NC)
61 ON R GOTO 63,113,91
63 REM
65 REM CREAZIONE FILE
67 GOSUB267:GOSUB275:GOSUB281
69 LC$="":LN$="":K1=N:FORK=1TON
71 GOSUB211
73 IFSW<>0THENK1=K-1:K=N:GOTO85
75 IFI$(1)>LC$THEN81
77 IFI$(1)=LC$THENIFI$(2)>LN$THEN81
79 GOSUB313:GOSUB319:GOTO71
81 LC$=I$(1):LN$=I$(2)
83 GOSUB221
85 NEXTK:CLOSE1
87 PRINT"FINITO CARICAMENTO   ";K1;" RECORD"
89 PRINT"FILE: ";NF$:GOSUB319:GOSUB365:RUN
91 REM
93 REM STAMPA FILE
95 GOSUB285:GOSUB267:GOSUB275:GOSUB293
97 PRINT#NF,"LISTA FILE ";NF$:PRINT#NF:PRINT#NF
99 GOSUB299
101 PRINT#NF,I$(1);SP$;I$(2)
103 PRINT#NF,I$(3);SP$;I$(4);SP$;I$(5);SP$;I$(6)
105 PRINT#NF:PRINT#NF
107 IFFS<>64THEN99
109 CLOSE1:CLOSENF:PRINT"FINITO LISTA"
111 GOSUB365:RUN
113 REM
115 REM AGGIORNAMENTO FILE
117 PRINT"AGGIORNAMENTO FILE"
119 PRINT"MONTA NASTRO VECCHIO"
121 GOSUB319:GOSUB275:GOSUB291:FORK=1TON
123 FORJ=1TONC:INPUT#1,MC$(K,J):NEXTJ
```

```
125 IFST=64THENCLOSE1:K1=K:K=N:NEXTK:GOTO129
127 NEXTK:PRINT"♥FILE SUPERA NUM.MASS.REC.";N:ST
OP
129 PRINT"♥LETTI ";K1;" RECORD":SW=0:N=K1
131 PRINT"♥♥VARIAZIONI S/N ":INPUTR$
133 IFR$<>"S"THEN149
135 REM
137 REM MODIFICA RECORD
139 SW=0:GOSUB375:IFSW<>0THEN149
141 SW=0:GOSUB387:IFSW<>0THEN135
143 FORJ=3TONC:I$(J)=MC$(K1,J):NEXTJ
145 GOSUB357:GOSUB337
147 FORJ=3TONC:MC$(K1,J)=I$(J):NEXTJ:GOTO135
149 PRINT"♥CANCELLAZIONI S/N ":INPUTR$
151 IFR$<>"S"THEN163
153 REM
155 REM CANCELLAZIONE RECORD
157 SW=0:GOSUB375:IFSW<>0THEN163
159 SW=0:GOSUB387:IFSW<>0THEN155
161 MC$(K1,1)=SP$:GOTO155
163 REM
165 REM RISCRITTURA FILE
167 SW=0:FF=0:K1=0:N1=1
169 PRINT"♥PREPARA NUOVO NASTRO":GOSUB267
171 GOSUB275:GOSUB281
173 PRINT"♥INSERIMENTI S/N ":INPUTR$
175 IFR$<>"S"THEN201
177 REM
179 REM INSERIMENTO NUOVO RECORD
181 GOSUB213:IFSW<>0THEN199
183 IFFF=1THEN189
185 GOSUB233
187 IFU<>0THENU=0:GOTO193
189 IFI$(1)>LC$GOTO195
191 IFI$(1)=LC$ANDI$(2)>LN$THEN195
193 GOSUB317:GOSUB319:GOTO181
195 GOSUB221:LC$=I$(1):LN$=I$(2)
197 K1=K1+1:GOTO181
199 IFFF=1THEN203
201 GOSUB253
203 PRINT"♥FINITO AGGIORNAMENTO"
205 PRINT"SCRITTI ";K1;" RECORD":CLOSE1
207 PRINT"FILE: ";NF$
209 GOSUB319:GOSUB365:RUN
211 REM
213 REM LETTURA NUOVI DATI
215 SW=0:GOSUB325:IFSW=1THENRETURN
217 GOSUB337
219 RETURN
221 REM
223 REM SCRIVE NUOVO RECORD SU NASTRO
225 FORJ=1TONC
227 IFLEN(I$(J))=0THENI$(J)=" "
```

```
229 PRINT#1,I$(J);CH$;:NEXTJ:RETURN
231 REM
233 REM RICERCA POSIZIONE NUOVO RECORD
235 U=0:FORK=N1TON:IFMC$(K,1)=SP$THEN249
237 IFMC$(K,1)<I$(1)THEN247
239 IFMC$(K,1)=I$(1)ANDMC$(K,2)<I$(2)THEN247
241 IFMC$(K,1)=I$(1)ANDMC$(K,2)=I$(2)THEN245
243 N1=K:K=N:NEXTK:RETURN
245 U=1:PRINT"DATI UGUALI":GOSUB319:GOTO243
247 GOSUB303:K1=K1+1:LC$=MC$(K,1):LN$=MC$(K,2)
249 NEXTK
251 FF=1:RETURN
253 REM
255 REM TERMINA SCRITTURA FILE
257 IFN1=NANDFF=1THENRETURN
259 FORK=N1TON:IFMC$(K,1)=SP$THEN263
261 GOSUB305:K1=K1+1
263 NEXTK:RETURN
265 REM
267 REM RICHIESTA PREPARAZIONE NASTRO
269 PRINT"QMONTA NASTRO":GOSUB319
271 RETURN
273 REM
275 REM RICHIESTA NOME FILE
277 INPUT"QNOME FILE ";NF$:RETURN
279 REM
281 REM APERTURA FILE PER SCRIVERE
283 OPEN1,1,2,NF$:RETURN
285 REM
287 REM APERTURA FILE DI STAMPA
289 OPENNF,PR:RETURN
291 REM
293 REM APERTURA FILE PER LEGGERE
295 OPEN1,1,0,NF$:RETURN
297 REM
299 REM LETTURA RECORD DA NASTRO
301 FORJ=1TONC:INPUT#1,I$(J):NEXTJ:FS=ST:RETURN
303 REM
305 REM SCRITTURA RECORD NASTRO DA VETTORI
307 FORJ=1TONC
309 IFLEN(MC$(K,J))=0THENMC$(K,J)=" "
311 PRINT#1,MC$(K,J);CH$;:NEXTJ:RETURN
313 REM
315 REM MESSAGGIO FUORI ORDINE
317 PRINT"QFUORI ORDINE "I$(1)SP$I$(2):RETURN
319 REM
321 GETKEYA$:IFA$=""THEN321:REM ATTESA TASTO
323 RETURN
325 REM
327 REM INGRESSO NUOVI DATI
329 FORJ=1TONC:I$(J)=" ":NEXTJ
331 PRINT"L";:J=1:GOSUB351
333 IFI$(1)="*"THENSW=1:RETURN
```

```
335 FORJ=2TONC:GOSUB351:NEXTJ:RETURN
337 REM
339 REM CONFERMA DATI E CORREZIONE
341 PRINT"{DOWN}CONFERMI S/N":INPUTR$
343 IFR$="S"THENRETURN
345 INPUT"QUALE CAMPO ";J
346 IFJ<1ORJ>NCTHEN345
347 PRINTD$(J)" ";:I$(J)=" ":INPUTI$(J)
349 GOSUB359:GOTO339
351 REM RICHIESTA CAMPO
353 PRINTJ;" ";D$(J);" ";:I$(J)=" "
355 INPUTI$(J):RETURN
357 REM
359 REM STAMPA DATI RECORD
361 PRINT"{CLR}";:FORJ=1TONC
363 PRINTJ;" ";D$(J);" ";I$(J):NEXTJ:RETURN
365 REM
367 REM ULTIMO MESSAGGIO
369 PRINT"{CLR}{DOWN}{DOWN}{DOWN}{DOWN}RIAVVOLGI IL NASTRO"
371 PRINT"SE NECESSARIO"
373 GOSUB319:RETURN
375 REM
377 REM RICHIESTA PRIMI DUE CAMPI
379 PRINT"RISPONDI * PER     USCIRE"
380 I$(1)="":I$(2)=""
381 PRINTD$(1);:INPUTI$(1)
383 IFI$(1)="*"THENSW=1:RETURN
385 PRINTD$(2);:INPUTI$(2):RETURN
387 REM
389 REM RICERCA RECORD
391 FORK=1TON
393 IFMC$(K,1)=I$(1)ANDMC$(K,2)=I$(2)THEN401
395 NEXTK:SW=1
397 PRINT"{DOWN}NON TROVATO ";I$(1);SP$;I$(2)
399 GOSUB319:RETURN
401 K1=K:K=N:NEXTK:RETURN
```

COMMENTO AL PROGRAMMA

Il programma e' organizzato in modo che dopo aver preso visione di un quadro video, devi premere un tasto per proseguire; la cosa e' realizzata con il sottoprogramma delle linee 319/323.

.1/5: commenti.

.7: PR e' il "dn" della periferica di stampa, noi abbiamo usato 3 per far uscire sul video; puoi sostituire il dn della stampante se ne disponi. Nel caso tu abbia una stampante, dovresti migliorare il programma introducendo un contatore per il numero delle linee di stampa, in modo da cambiare foglio al momento giusto. Per leggere piu' comodamente sul video i dati, durante la stampa ti conviene tenere premuto il tasto CBM LOGO.

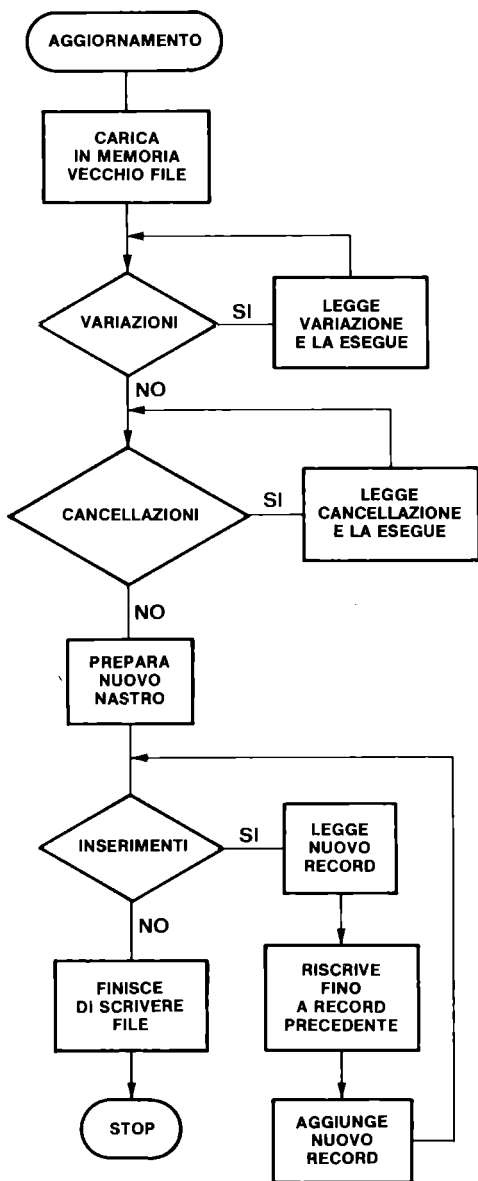



Figura 4.3 Fase AGGIORNAMENTO di FILESEQ

.9: NF e' il numero logico del file di stampa, lo "lfn"; noi abbiamo usato 4. In realta' per fare uscire i dati sul video potevamo evitare di aprirlo come file, lo abbiamo fatto per rendere piu' semplice il trasferimento dell'uscita su una stampante.

.11: NC e' il numero dei campi del record; viene usato per definire le variabili con indice e per gestire i cicli.

.13/15: dimensionamento del vettore di stringhe D$ che deve contenere le descrizioni dei campi del record, e del vettore di stringhe I$, che deve contenere i dati da scrivere nel record, quando questi vengono letti dalla tastiera.

.17: CH$ contiene il carattere RETURN, da usare come separatore nella scrittura dei campi; usiamo sempre CHR$(13) per non avere problemi con la lunghezza del record. SP$ contiene 3 spazi.

.19/25: le due frasi DATA contengono le descrizioni dei campi, che vengono trasferite nel vettore D$ con la linea 25. Se si aumenta il numero dei campi NC, si devono aggiungere altri dati.

.27/41: presentazione del menu' principale per scegliere la funzione da svolgere. La linea 39 accetta la risposta, la 41 la controlla e accetta solo: 1, 2, 3 e 9.

.43: se R=9 (FINE) viene eseguita la subroutine che chiede di riavvolgere il nastro se necessario e il programma si ferma.

.45/55: viene chiesto il numero totale di record da elaborare, N. Esso serve per predisporre la zona per memorizzare il file in caso di aggiornamento. Questo dato e' importante e quindi ne viene chiesta conferma.

.57/59: viene dimensionata la matrice di stringhe MC$, di N righe e NC colonne per contenere il file, se si devono operare aggiornamenti.

.61: scelta della funzione principale in base al valore di R.

.63/89: fase di CREAZIONE ex novo del file; essa si esegue la prima volta. Alla 67 vengono eseguiti i sottoprogrammi per preparare il nastro, richiedere il nome del file e aprire il file in scrittura. Alla 69 vengono poste a stringa nulla le due variabili LC$ e LN$, che vengono usate per controllare la sequenza dei record caricati, in base ai primi due campi. Poi inizia un ciclo, da percorrere al massimo N volte, per caricare i record richiedendo i dati dalla tastiera. Il sottoprogramma in 211 legge i nuovi dati dalla tastiera. Se ritorna con SW<>0, questo significa che il caricamento e' terminato, allora viene conservato in K1 il numero dei record scritti, che puo' anche essere inferiore ad N, e si va alla fine di questa fase. Se e' stato caricato un nuovo record, ne viene controllata la sequenza rispetto al precedente; se va bene il record viene scritto con il sottoprogramma in 221 e viene chiesto un nuovo record. Se la sequenza non va bene questo viene segnalato; il record non viene accettato e potra' essere aggiunto in una successiva fase di aggiornamento del file. In 87 e 89 viene segnalato quanti record sono stati caricati e in

quale file e il programma viene rilanciato con RUN.

.91/111: fase di STAMPA. Alla 95 viene aperto il file di stampa, richiesto di montare il nastro, richiesto il nome del file e aperto per leggere. Poi vengono letti i record uno alla volta con il sottoprogramma in 297, stampati ponendo sulla prima riga i primi due campi e sulla seconda gli altri quattro. Nel caso tu aumenti il numero dei campi, dovrai probabilmente modificare le istruzioni alle linee 103 e 105. Viene controllata la fine del file analizzando la variabile riservata di stato ST, trasferita al momento della lettura in FS, e, quando il file e' terminato, la fase si arresta e dopo i soliti messaggi viene rilanciato il programma con RUN.

.113/129: inizia la fase di AGGIORNAMENTO. Come prima cosa viene trasferito in memoria, nella matrice MC$, il file, di cui e' stato chiesto il montaggio, il nome, ed e' stato aperto in lettura. Viene segnalato se il file e' piu' lungo del previsto numero di record, e in questo caso si ha uno STOP e ti consigliamo di ripartire dando per N un numero adeguato. Alla fine del caricamento viene segnalato il numero effettivo dei record letti. L'aggiornamento avviene in fasi successive; prima l'eventuale modifica dei record presenti in memoria, poi l'eventuale cancellazione di qualche record, e poi il caricamento dei nuovi record, durante la fase di riscrittura del file.

.131/147: fase MODIFICA record. Se non ci sono modifiche passa alla fase seguente. Alla 139 chiede i primi due campi del record da modificare con il sottoprogramma in 375; se al ritorno SW<>0, significa che non ci sono piu' modifiche. Alla 141 ricerca in memoria il record voluto, se non lo trova SW<>0 e ne chiede un altro. Legge dalla memoria gli altri campi del record selezionato e propone sul video tutto il record chiedendo quali campi vuoi modificare. Non devi modificare i primi due campi, perche' se lo fai alteri l'ordine dei record. Il programma riscrive in memoria solo i campi dal terzo in avanti, quindi anche se tu hai modificato i primi due, essi non vengono memorizzati. Per modificare i primi due campi devi cancellare il record e poi riscriverlo.

.149/161: fase CANCELLAZIONE. Come nella fase precedente vengono chiesti i primi due campi per individuare il record ed esso viene ricercato. Se esso viene trovato, vengono scritti degli spazi al posto del primo campo; in fase di riscrittura, se il primo campo contiene spazi, il record non viene scritto.

.163/171: viene predisposta la riscrittura del file su nastro, chiedendo di montare il nastro e quale nome registrare in apertura. Alla 167 vengono preparati i seguenti indicatori:

. SW=0, se diventa 1 significa che non ci sono piu' nuovi record da aggiungere,

. FF=0, se diventa 1 significa che sono stati gia' scritti tutti i record della matrice MC$,

. K1=0, contatore dei record che si scrivono,
. N1=1, puntatore alla posizione del record nella matrice MC$.
.173/175: richiesta se ci sono inserimenti.
.177/209: fase INSERIMENTO e RISCRITTURA. Alla 181 viene chie¬sto un nuovo record; se viene fornito, vengono scritti sul nuovo file tutti i record di MC$, che precedono nell'ordinamento il nuovo record, e poi viene scritto il nuovo record. Non sono accettati record doppi. Quando MC$ e' esaurito si procede solo con nuove acquisizioni; quando i nuovi record sono terminati si esaurisce MC$, se FF=0. La fase di riscrittura e' la piu' compli¬cata di tutto il programma, dato che vogliamo mantenere l'ordi¬namento dei record. Alla fine viene segnalato quanti record si sono scritti e il nome del file.
.211/219: sottoprogramma lettura nuovi dati da tastiera, serven¬dosi dei sottoprogrammi in 325 e 337.
.221/229: scrittura nuovo record su nastro; i dati sono quelli ricevuti da tastiera. Ti facciamo notare che se un dato stringa ha lunghezza zero, esso viene sostituito con una stringa di uno spazio. Questo e' necessario perche' sul nastro venga registrato un campo valido; la stringa nulla viene saltata in fase di let¬tura e produce uno scompenso nella struttura dei record, ognuno dei quali deve avere un numero prefissato di campi. Ricorda che se leggi da tastiera una stringa formata da uno o piu' spazi, es¬sa viene conservata come stringa nulla; questo non succede se usi SHIFT¬spazio. Se, invece scrivi A$=" ", la stringa A$ ha lunghez-za 1 e quando la scrivi su nastro lo spazio viene registrato e conta come un campo.
.231/251: ricerca posizione per il nuovo record in MC$. Viene segnalato se i dati sono fuori ordine o sono doppi. Se tutto be¬ne vengono trasferiti sul file i record di MC$ che precedono quello nuovo.
.253/263: viene portata su file l'ultima parte di MC$.
.265/271: richiesta di preparazione nastro.
.273/277: richiesta nome file.
.279/283: apertura file per scrivere.
.285/289: apertura file di stampa.
.291/295: apertura file per leggere.
.297/301: lettura record da nastro.
.303/311: scrittura record prelevato da matrice MC$, con modi¬fica eventuali stringhe nulle.
.313/317: messaggio record fuori ordine.
.319/323: attesa pressione tasto per proseguire.
.325/335: ingresso nuovi dati da tastiera; accetta anche campi vuoti.
.337/349: visualizzazione dati, richiesta conferma o corre¬zione.
.351/355: richiesta di un campo.
.357/363: visualizzazione dati di un record.

.365/373: messaggio finale di ogni fase.
.375/385: richiesta dati primi due campi, pone SW=1 se rispondi con * al primo campo.
.387/401: ricerca record in MC$.

Nel programma FILESEQ abbiamo usato la tecnica dei sottoprogrammi, cioe' abbiamo scritto sotto forma di sottoprogramma tutti i gruppi di istruzioni che devono essere eseguite piu' di una volta; questa tecnica, oltre che portare a un risparmio di memoria, evita errori di programmazione.

Riportiamo l'elenco delle variabili usate nel programma:

PR=3, dn periferica stampa
NF=4, lfn file di stampa
NC=6, numero campi del record
D$(NC), descrizioni campi record
I$(NC), dati del record
CH$=CHR$(13), carattere separatore RETURN
SP$, costante contenente 3 spazi
R$, stringa per risposte
R, numero della scelta
N, numero record
K, variabile controllo ciclo
MC$(N,NC), matrice per contenere i record in memoria
LC$, primo campo del record precedente
LN$, secondo campo del record precedente
SW, flag per controllare se e' stato fornito un nuovo record
K1, numero record caricati
NF$, nome del file
FS, per conservare la parola di stato ST
ST, parola di stato
J, variabile controllo cicli
FF, controllo per record MC$
N1, puntatore a MC$
U, controllo record da inserire uguali a record MC$

La gestione del programma e' molto semplice:
.vengono evidenziati messaggi di richiesta sul video e devi rispondere,
.quando il quadro video e' fisso, e la cassetta non e' attiva, per proseguire devi premere un tasto,
.per uscire dalle fasi di richiesta dati devi rispondere con * al primo dato,
.devi stare attento al cambiamento del nastro o al suo riavvolgimento per non perdere registrazioni.

Per adattare FILESEQ alle tue esigenze puoi apportare le seguenti modifiche:

. porre PR a un nuovo valore, per usare una stampante,
. modificare NC e modificare le linee DATA per le descrizioni dei campi.
Resta comunque un programma che mantiene i record in ordine in base ai primi due campi; questa e' una cosa piu' difficilmente modificabile, potresti pero' mantenere uno dei campi sempre uguale a SHIFT-spazio e quindi ordinare solo su uno.
Inoltre FILESEQ non accetta record doppi.
Puoi aggiungere al programma altre parti, per ottenere altre funzioni, e puoi fare uso di alcuni dei sottoprogrammi di uso generale gia' presenti.
Il programma FILESEQ lascia liberi 56201 byte per le variabili; devi fare un po' di conti per decidere il massimo numero di record gestibili con MC$.

4.5 I JOYSTICK

Il BASIC ci mette a disposizione un'istruzione per "leggere" lo stato dei Joystick: JOY(n), con n che puo' essere 1 o 2, e corrisponde alle prese JOY 1 e JOY 2.
Questa istruzione puo' essere usata per trasferire lo stato attuale del Joystick in una variabile, scrivendo, per esempio, A=JOY(1). Oppure direttamente, per analizzare in tempo reale lo stato del Joystick, per esempio cosi': IF JOY(1)=1 THEN....
I valori forniti dall'istruzione JOY(n) sono 9 + 9; essi corrispondono agli 8 movimenti possibili + lo stato di riposo, ottenuti con il solo movimento della leva, e agli 8 movimenti possibili + lo stato di riposo con l'aggiunta del fuoco, ottenuti premendo il bottone rosso mentre si muove la leva. Lo schema che segue mostra le due situazioni possibili.

SOLO MOVIMENTO   MOVIMENTO + FUOCO

Per utilizzare i joystick come periferiche di Input e' necessario leggere il loro stato e produrre in conseguenza sul video delle azioni, che in generale sono di movimento, ma potrebbero essere di qualunque tipo, infatti dipendono dal programma.

Abbiamo preparato il programma JOYSTICK1 per dimostrare la gestione dei joystick in BASIC; esso gestisce il joystick 1. In esso abbiamo creato dei cicli di attesa per consentire di analiz-

```
118 ONAGOTO124,127,130,133,136,139,142,145
121 STOP
124 PRINT"";C3$;R4$;"";S$;:GOTO109
127 PRINT"";C3$;R5$;"";S$;:GOTO109
130 PRINT"";C4$;R5$;"";S$;:GOTO109
133 PRINT"";C5$;R5$;"";S$;:GOTO109
136 PRINT"";C5$;R4$;"";S$;:GOTO109
139 PRINT"";C5$;R3$;"";S$;:GOTO109
142 PRINT"";C4$;R3$;"";S$;:GOTO109
145 PRINT"";C3$;R3$;"";S$;:GOTO109
148 REM SOLO FUOCO
151 PRINT"";C4$;R4$;"";S$;:GOTO109
```

COMMENTO A JOYSTICK1

.1/25: prepara stringhe di caratteri di controllo per spostarsi sul video verso il basso e verso destra. R3$ manda a destra di 13 posizioni. R4$ manda a destra di 14 posizioni. R5$ manda a destra di 15 posizioni. C3$ manda in giu' di 13 posizioni. C4$ manda in giu' di 14 posizioni. C5$ manda in giu' di 15 posizioni. M$ serve per numerare le colonne del video.

.28/31: S$ serve per visualizzare uno spazio in campo inverso, N$ per visualizzare uno spazio e quindi cancellare quanto presente in quella posizione.

.34/46: numera le righe e le colonne del video.

.49/52: visualizza un quadratino nero nella posizione 14,14.

.55/64: legge lo stato del joystick 1 fino a quando lo trova diverso da 0, crea un ciclo di attesa, e cancella il quadratino dalla vecchia posizione.

.67/70: se stato=128 va alla linea 148; se stato>128 va alla linea 112.

.73/103: in base al valore letto esegue il movimento, cioe' disegna il quadratino nella nuova posizione in nero.

106/109: crea un ciclo di attesa e poi torna alla linea 37 per ricominciare.

112/145: in base al valore letto esegue il movimento, cioe' disegna il quadratino nella nuova posizione in rosso, per segnalare che e' stato premuto anche il bottone del fuoco.

148/151: disegna nella vecchia posizione il quadratino in rosso, per segnalare solo bottone del fuoco.

Abbiamo preparato il programma JOYSTICK2 per dimostrare la gestione dei joystick in BASIC con video grafico; esso lavora con il joystick 1. In esso abbiamo preparato un piccolo rombo e lo abbiamo memorizzato come SHAPE nella stringa D$. Lo stato del joystick 1 viene usato per spostare lo SHAPE sul video. Anche in questo caso abbiamo creato dei cicli di attesa per consentire di studiare le caratteristiche dei joystick. Se viene premuto anche il bottone del fuoco il rombo viene colorato di nero nella vec-

chia posizione, poi viene cancellato e disegnato nella nuova posizione. La posizione iniziale e' X1=154, Y1=99. Gli incrementi per X1 e Y1 in base ai movimenti (primo numero per X, secondo per Y) sono i seguenti:

```
-10,-10     0,-10     10,-10

-10, 0                10, 0

-10, 10     0, 10     10, 10
```

```
1 REM JOYSTICK2
4 REM ATTIVA MODO GRAFICO
7 GRAPHIC1,1
10 REM PREPARA SHAPE, PICCOLO ROMBO
13 X=159:Y=99
16 DRAW1,X,Y TO X+5,Y+5 TO X,Y+10
19 DRAW1 TO X-5,Y+5 TO X,Y
22 SSHAPED$,X-5,Y,X+5,Y+10
25 X1=X-5:Y1=Y
28 REM INTERROGA JOYSTICK 1
31 A=JOY(1)
34 IFA=0THEN31
37 IFA=128THEN85
40 IFA>129THEN97
43 REM USO JOYSTICK SENZA FUOCO
46 REM XD E YD INCREMENTI COORDINATE
49 IFA=1THENXD=0:YD=-10:GOTO76
52 IFA=2THENXD=10:YD=-10:GOTO76
55 IFA=3THENXD=10:YD=0:GOTO76
58 IFA=4THENXD=10:YD=10:GOTO76
61 IFA=5THENXD=0:YD=10:GOTO76
64 IFA=6THENXD=-10:YD=10:GOTO76
67 IFA=7THENXD=-10:YD=0:GOTO76
70 IFA=8THENXD=-10:YD=-10:GOTO76
73 REM CANCELLA VECCHIO E DISEGNA NUOVO
76 FORK=1TO300:NEXTK:GSHAPED$,X1,Y1,4
79 X1=X1+XD:Y1=Y1+YD
82 GSHAPED$,X1,Y1:FORK=1TO300:NEXTK:GOTO31
85 REM SOLO FUOCO, COLORA VECCHIO
88 REM CANCELLA E DISEGNA VECCHIO
91 PAINT,X1+5,Y1+5:FORK=1TO300:NEXTK
94 GSHAPED$,X1,Y1:GOTO31
97 REM MOVIMENTO + FUOCO
100 REM COLORA VECCHIO E POI CANCELLA
103 REM DISEGNA NUOVO
106 PAINT,X1+5,Y1+5:FORK=1TO300:NEXTK
109 GSHAPED$,X1,Y1
112 IFA=129THENXD=0:YD=-10:GOTO76
115 IFA=130THENXD=10:YD=-10:GOTO76
118 IFA=131THENXD=10:YD=0:GOTO76
121 IFA=132THENXD=10:YD=10:GOTO76
124 IFA=133THENXD=0:YD=10:GOTO76
```

```
127 IFA=134THENXD=-10:YD=10:GOTO76
130 IFA=135THENXD=-10:YD=0:GOTO76
133 IFA=136THENXD=-10:YD=-10:GOTO76
```

COMMENTO A JOYSTICK2

1/7: passa in modo grafico e cancella il video.

10/25: prepara il disegno e lo memorizza. Le coordinate della prima posizione sono X1=154 e Y1=99, angolo in alto a sinistra del rombo.

28/40: memorizza lo stato del joystick 1 e sceglie in base al valore da dove proseguire.

43/70: caso solo movimento: in base al valore letto prepara XD e YD, incrementi per X1 e Y1, per definire la nuova posizione e prosegue.

73/82: crea un ciclo di attesa, cancella vecchio disegno, incrementa X1 e Y1 e disegna nella nuova posizione, poi torna a leggere lo stato del joystick 1.

85/94: caso solo fuoco: colora il rombo nella vecchia posizione, crea un ciclo di attesa, lo cancella e lo ridisegna.

97/133: caso movimento + fuoco: colora il rombo nella vecchia posizione, crea un ciclo di attesa, ridisegna nella vecchia posizione, poi prepara XD e YD, e va alla linea 76.

CAPITOLO 5

I FILE SU STAMPANTE

5.1 INTRODUZIONE

In questo capitolo descriviamo l'uso della stampante COMMODORE MPS-803, che viene di norma venduta per il calcolatore COMMODORE PLUS-4. Il calcolatore puo' essere collegato anche ad altre stampanti senza problemi; alcune si possono collegare direttamente, per altre e' necessaria un'interfaccia speciale. La maggior parte dei discorsi che facciamo restano validi anche per altri tipi di stampanti; nel caso devi chiederne al venditore le caratteristiche, vedere se e' necessaria un'interfaccia e leggere accuratamente il manuale allegato per scoprire le differenze.
La MPS-803 e' una stampante a impatto a matrice di punti, nella quale un carattere e' formato da 6 punti orizzontali e 7 punti verticali. Essa riconosce e stampa tutti i caratteri dei due set disponibili sul COMMODORE PLUS-4: maiuscolo/grafico e minuscolo/maiuscolo. Inoltre puo' stampare colonne di punti (7 punti verticali) e quindi e' una stampante grafica; la colonna di punti deve essere opportunamente codificata.
Essa si collega tramite l'interfaccia seriale standard al COMMODORE PLUS-4, mediante il cavo fornito che termina con uno spinotto DIN a 6-pin.
Sulla parte superiore della MPS-803 sono visibili:
 .in alto a sinistra, la manopola per l'avanzamento manuale della carta;
 .in alto a destra, la levetta per il bloccaggio della carta (da usare quando non e' montato il trascina moduli);
 .in basso a destra, una spia luminosa rossa, marcata Power, che segnala:
   .con luce continua che la macchina e' accesa,
   .con luce intermittente che si e' verificato un errore,
e vicino un tasto a pressione, marcato Paper Advance, per l'avanzamento della carta.
La copertura anteriore puo' essere sollevata, agendo su due appositi incavi laterali. Per inserire il nastro inchiostrato si deve sollevare tale copertura e si rende visibile l'alloggiamento del nastro. Nel manuale allegato alla stampante sono riportate

delle illustrazioni che insegnano a montare il nastro.
Sul lato destro in basso si trova l'interruttore di accensione.
La stampante puo' essere alimentata con moduli continui o con fogli singoli, usando la levetta di bloccaggio della carta (quando e' in posizione OPEN la carta e' libera); oppure puo' essere montato il trascina moduli, richiedendolo al venditore. La larghezza della linea di stampa e' di 80 caratteri (larghi 6 punti), quindi di 480 punti. Possiamo assumere la larghezza della linea di stampa come dimensione del record fisico. La stampa e' bidirezionale e la velocita' di stampa e' di 60 caratteri al secondo. Si possono stampare fino a 3 copie. La stampante non puo' funzionare se la carta non e' inserita. Quando, durante la stampa, termina la carta, comincia a pulsare la spia rossa Power e la stampa si interrompe; per proseguire devi inserire nuova carta e premere il tasto Paper Advance.
Lo switch posteriore DEVICE puo' essere posto nella posizione 4 o nella posizione 5, dando la possibilita' di scegliere il "dn" della stampante. Di solito si usa il 4, ma, se sono collegate contemporaneamente due stampanti, una deve avere dn=5.
Lo switch posteriore LPI puo' essere posto nelle posizioni 6 o 8; esso indica la distanza di 1/6 o 1/8 di pollice tra due linee di stampa.
La porta seriale SERIAL INTERFACE dispone di due ingressi; uno serve per collegare la stampante al calcolatore, l'altro per collegare in cascata una seconda stampante o una unita' a floppy disk. Si possono collegare in cascata piu' periferiche (vedi Paragrafo 7.10).

Il COMMODORE PLUS-4 puo' inviare dati alla stampante dopo avere stabilito una comunicazione con essa; la stampante riceve un flusso di dati e stampa un file. La MPS-803 contiene un microprocessore che gestisce le operazioni di stampa, eseguendo apposite routine memorizzate in una ROM interna, servendosi di una descrizione dei caratteri memorizzata in una ROM interna, e di un buffer di stampa, cioe' di una parte di RAM interna.
Il COMMODORE PLUS-4 invia i dati, attraverso l'interfaccia seriale, un bit dopo l'altro; ogni gruppo di 8 bit (un byte) contiene un codice ASCII, che puo' variare da 0 a 255. I byte sono ricevuti e memorizzati uno dopo l'altro nel buffer, che puo' contenere fino a 90 byte; vedremo piu' avanti in quali circostanze avviene realmente la stampa.
Alcuni dei codici inviati non sono caratteri da stampare, ma codici di controllo che agiscono sul modo di funzionare della stampante.
Quando la stampante funziona, al momento dell'accensione, sia che sia collegata al calcolatore, sia che non lo sia, la testina di stampa viene spostata dal centro verso sinistra e poi riportata al centro. Per vedere se la stampa e' corretta puoi farle esegui-

re l'auto-test. Devi procedere cosi':
.aver montato correttamente il nastro e inserito la carta,
.dare corrente alla stampante, anche non collegata al calcolatore, mentre tieni premuto il tasto anteriore "Paper Advance", poi rilasciare tale tasto.
La stampante stampa ripetutamente tutti i caratteri stampabili.
Per interrompere il test devi togliere corrente.

5.2 ISTRUZIONI DI STAMPA

Prima di stampare si deve aprire la comunicazione tra il calcolatore e la stampante con l'istruzione OPEN; essa si scrive:

```
OPEN lfn,dn,sa
```

.lfn (Logical File Number), e' il numero logico del file che si vuole aprire e che deve essere usato nelle istruzioni di stampa successive; esso puo' variare da 0 a 255. Se lfn>127 si ottiene una spaziatura doppia tra una linea e la successiva.
.dn (Device Number), e' il numero della periferica, puo' essere 4 o 5, e deve essere quello predisposto dall'apposito switch posto sul retro della stampante.
.sa (Secondary Address), e' un numero che puo' valere 0 (valore di default) o 7 e determina il set di caratteri che la stampante deve usare nelle successive operazioni di stampa:
.0, o niente, per il set maiuscolo/grafico,
.7, per il set minuscolo/maiuscolo.
I parametri possono essere costanti o variabili con valore intero.
Il set di caratteri scelto con la OPEN resta attivo fino alla CLOSE; si puo' usare all'interno della lista di stampa un codice di controllo che modifica temporaneamente il set, ma esso e' valido solo per la stampa in corso, cioe' fino al primo carattere RETURN.

L'istruzione di stampa e':

```
PRINT# lfn,lista
```

.lfn, deve essere lo stesso usato nella OPEN.
.lista, e' l'insieme dei dati da stampare, dei caratteri separatori, delle funzioni di stampa e dei codici di controllo per la stampante. Negli esempi successivi vedremo le caratteristiche di ogni elemento che puo' comparire nella lista.
Per scrivere questa istruzione non si puo' usare il "?" per abbreviare la parola chiave, inoltre non deve esserci spazio prima del carattere "#".

Il calcolatore, dopo aver eseguito un'istruzione PRINT#, chiude la comunicazione con la stampante, ma il file logico rimane aperto; esso viene chiuso solo dall'istruzione CLOSE.

Si puo' stampare su carta, anche usando l'istruzione CMD, dopo aver aperto un file per la stampante. Questa istruzione trasferisce l'uscita, che normalmente avviene sul video, alla stampante. Si scrive:

```
CMD lfn,lista
```

.lfn, deve essere lo stesso usato nell'istruzione OPEN.
.lista, puo' essere una normale lista di stampa o mancare.

Dopo l'esecuzione di CMD, con o senza "lista", le successive istruzioni PRINT (scritte senza il carattere "#") mandano l'output sulla stampante. Per far terminare l'effetto di CMD, cioe' il "dirottamento" dell'uscita dal video a un'altra periferica, e' necessario eseguire una PRINT#lfn, con "lfn" uguale a quello usato per CMD, che serve per chiudere la linea, prima della CLOSE. In caso contrario non si ha un funzionamento corretto del sistema; inoltre, quando si e' in stato CMD, non si deve accedere a un'altra periferica collegata in serie (disco o altra stampante). Il comando LIST, usato dopo CMD, provoca la lista del programma sulla stampante.
Il modo corretto per ottenere la lista dei programmi sulla stampante e' eseguire in immediato:

```
OPEN4,4:CMD4:LIST:PRINT#4:CLOSE4
```

puo' essere utile assegnare a un tasto funzione la sequenza di istruzioni necessarie per listare i programmi, eseguendo per esempio:
KEY1,"OPEN4,4:CMD4:LIST:PRINT#4:CLOSE4"+CHR$(13)
in modo che premendo F1 si ottiene la lista con chiusura corretta del file.
Quando le operazioni di stampa sono terminate, deve essere eseguita l'istruzione:

```
CLOSE lfn
```

che serve per chiudere la comunicazione. Dopo l'esecuzione della CLOSE non si puo' piu' comunicare con il file "lfn", se non si esegue nuovamente una OPEN.

E' importante non lasciare aperti i file che non servono piu'; si rischia di occupare inutilmente posto nella tabella dei file, che ha solo 10 posti.

Le istruzioni di stampa possono essere usate sia in modo immediato che da programma.

ESEMPI DI STAMPA IN MODO IMMEDIATO

Usando in modo immediato le 4 sequenze di istruzioni che seguono, si ottiene sempre lo stesso risultato sulla stampante e si opera correttamente chiudendo sia la linea che il file logico.

.1)
OPEN4,4:PRINT#4,"MPS-803":CLOSE4
stampa MPS-803 e va a capo.
.2)
OPEN4,4:CMD4,"MPS-803":PRINT#4:CLOSE4
stampa MPS-803 e va a capo.
.3)
OPEN4,4:CMD4:PRINT"MPS-803":PRINT#4:CLOSE4
stampa MPS-803 e va a capo.
.4)
OPEN4,4,7:CMD4:PRINT"MPS-803":PRINT#4:CLOSE4
stampa mps-803 (abbiamo usato sa=7) e va a capo.

ESEMPI DI STAMPA DA PROGRAMMA

Il programma SAOCMDLIST, che segue con i suoi risultati, apre il file logico con lfn=1 sulla stampante (dn=4), usando sa=0, che poteva anche essere saltato, dato che 0 e' il valore di default, e quindi lavora con il set maiuscolo/grafico. Nel programma alla linea 15 e' inserito il comando LIST, cioe' il programma lista se stesso, poi continua e stampa una frase. Alla linea 25 viene eseguita l'istruzione PRINT#1 per chiudere la linea e poi la CLOSE1.

```
1 REM SA0CMDLIST
5 OPEN1,4,0:REM APRE CON SA=0
10 CMD1:REM USCITA DA VIDEO A STAMPANTE
15 LIST:PRINT
20 PRINT"OPEN CON SA=0 E USO CMD E LIST"
25 PRINT#1:CLOSE1

OPEN CON SA=0 E USO CMD E LIST
```

Il programma SA7CMDLIST e' uguale al precedente, salvo che apre il file di stampa con sa=7 e quindi lavora con il set di caratteri minuscolo/maiuscolo.

Segue il programma SETMPS¬803, che stampa in forma tabellare i due set di caratteri disponibili sulla stampante. Il disegno dei caratteri e' quello registrato nella ROM interna alla stampante, che differisce un po' dalla forma dei caratteri che appare sul video: 6x7 punti per la stampante, 8x8 punti sul video.

```
1 rem sa7cmdlist
5 open1,4,7:rem apre con sa=7
10 cmd1:rem uscita da video a stampante
15 list:print
20 print"open con sa=7 e uso cmd e list"
25 print#1:close1

open con sa=7 e uso cmd e list
```

I caratteri da stampare sono inviati come codici ASCII; il programma stampa spazi al posto dei caratteri corrispondenti ai codici da 0 a 31 e da 128 a 159, formando le due colonne vuote nelle tabelle, dato che non corrispondono a caratteri stampa¬bili. Le coordinate orizzontali e verticali delle due tabelle so¬no espresse in esadecimale (le cifre esadecimali da A a F corrispondono ai numeri decimali da 10 a 15). Per ottenere il codice ASCII dei caratteri devi moltiplicare per 16 la coordi¬nata di colonna (sopra) e aggiungere la coordinata di riga (late¬rale).

```
1 REM SETMPS-803
5 A$="     SET MAIUSCOLO/GRAFICO"
6 SA=0
10 OPEN4,4,SA:PRINT#4,A$
11 PRINT#4:PRINT#4," | ";
12 FORK=0TO9:PRINT#4,MID$(STR$(K),2);" ";
13 NEXTK
14 FORK=65TO70:PRINT#4,CHR$(K);" ";:NEXTK
15 PRINT#4:PRINT#4,CHR$(192)"┤";
16 FORK=1TO16:PRINT#4,CHR$(192)CHR$(192);
17 NEXTK:PRINT#4:I=0
19 FORK=0TO9:PRINT#4,MID$(STR$(K),2);"| ";
20 GOSUB100:I=I+1:NEXTK
23 FORK=65TO70:PRINT#4,CHR$(K);"| ";
25 GOSUB100:I=I+1:NEXTK
27 IFSA=7THENCLOSE4:STOP
29 PRINT#4:PRINT#4:SA=7
30 A$="     SET MAIUSCOLO/MINUSCOLO"
31 CLOSE4:GOTO10
100 IK=I:FORL=1TO16
101 IFIK<32THENPRINT#4,"  ";:GOTO104
102 IFIK>127ANDIK<160THENPRINT#4,"  ";:GOTO104
103 PRINT#4,CHR$(IK);" ";
104 IK=IK+16:NEXTL:PRINT#4:RETURN
```

RISULTATI PROGRAMMA SETMPS-803

SET MAIUSCOLO/GRAFICO

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0				0	@	P	─	┐				┌	─	┐		┌
1			!	1	A	Q	♠	●			▌	┴	♠	●	▌	┴
2			"	2	B	R	│	─			▄	┬	│	─	▄	┬
3			#	3	C	S	─	♥			▔	┤	─	♥	▔	┤
4			$	4	D	T	─	│			▁	│	─	│	▁	│
5			%	5	E	U	─	╭			▏	▌	─	╭	▏	▌
6			&	6	F	V	─	╳			▒	▐	─	╳	▒	▐
7			'	7	G	W	│	○			▕	▔	│	○	▕	▔
8			(	8	H	X	│	♣			▒	▀	│	♣	▒	▀
9			)	9	I	Y	╮	│			◤	▄	╮	│	◤	▄
A			*	:	J	Z	╰	♦			▕	┘	╰	♦	▕	┘
B			+	;	K	[	╯	┼			├	▗	╯	┼	├	▗
C			,	<	L	£	└	▒			▖	▝	└	▒	▖	▝
D			-	=	M	]	╲	│			└	┘	╲	│	└	┘
E			.	>	N	↑	╱	π			┐	▘	╱	π	┐	▘
F			/	?	O	←	┌	◥			▂	▚	┌	◥	▂	π

set maiuscolo/minuscolo

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0				0	@	p	─	P				┌	─	P		┌
1			!	1	a	q	A	Q			▌	┴	A	Q	▌	┴
2			"	2	b	r	B	R			▄	┬	B	R	▄	┬
3			#	3	c	s	C	S			▔	┤	C	S	▔	┤
4			$	4	d	t	D	T			▁	│	D	T	▁	│
5			%	5	e	u	E	U			▏	▌	E	U	▏	▌
6			&	6	f	v	F	V			▒	▐	F	V	▒	▐
7			'	7	g	w	G	W			▕	▔	G	W	▕	▔
8			(	8	h	x	H	X			▒	▀	H	X	▒	▀
9			)	9	i	y	I	Y			▒	▄	I	Y	▒	▄
a			*	:	j	z	J	Z			▕	✓	J	Z	▕	✓
b			+	;	k	[	K	┼			├	▗	K	┼	├	▗
c			,	<	l	£	L	▒			▖	▝	L	▒	▖	▝
d			-	=	m	]	M	│			└	┘	M	│	└	┘
e			.	>	n	↑	N	▚			┐	▘	N	▚	┐	▘
f			/	?	o	←	O	▒			▂	▚	O	▒	▂	▚

COMMENTO A SETMPS-803

.5/6: prepara in A$ l'intestazione della prima tabella e pone SA=0.

.10: apre il file logico con lfn=4, il valore attuale di SA per la stampante, e stampa il titolo della tabellina.

.11/17: stampa le coordinate di colonna e le linee separatrici (CHR$(192) da' luogo a un trattino), e pone il contatore I a 0.

.19/20: stampa le linee corrispondenti alle coordinate di riga da 0 a 9, servendosi del sottoprogramma in 100 per stampare i caratteri del set attivo.

.23/25: come sopra per le linee con cordinate da A a F.
.27: se SA=7 chiude il file logico e si ferma.
.29/31: pone SA=7, prepara la nuova intestazione in A$, chiude il file logico e torna alla linea 10 per stampare la seconda tabellina.
.100/104: sottoprogramma che in base al codice del carattere, che trova inizialmente in I e pone in IK, stampa, se sono stampabili, i 16 caratteri di una linea. Non stampa il carattere se il codice e' minore di 32 oppure compreso tra 128 e 159, estremi inclusi. I caratteri che stanno su una linea hanno codici che differiscono di 16 da quello precedente.

5.3 MODI DI STAMPA E USO DEI CARATTERI DI CONTROLLO

Il valore di "sa" usato nella OPEN predispone l'uso di uno dei due set di caratteri disponibili nella MPS-803, in modo permanente, fino all'esecuzione della relativa CLOSE. Il valore di "sa" influenza la stampa in MODO CARATTERE, cioe' quella attiva al momento dell'accensione della stampante. Per ottenere il passaggio al MODO GRAFICO devi inviare un carattere di controllo (CHR$(8)); in tale caso e' necessario inviare un altro carattere di controllo per tornare al modo testo (CHR$(15)).
Possiamo considerare come una prima distinzione nelle possibilita' di stampa i due modi: carattere e grafico; ad essi possiamo aggiungere gli altri modi che si ottengono con l'uso di alcuni caratteri di controllo. Segue l'elenco dei caratteri di controllo che hanno un particolare significato per la stampante MPS-803; essi devono essere inviati nella "lista" di un'istruzione PRINT diretta alla stampante, come stringa, usando la funzione CHR$. Per ogni carattere indichiamo: il codice, il significato, i parametri che devono seguire il codice, se necessario, e il numero dei byte che vengono occupati in conseguenza nel buffer della stampante. Se un codice richiede dei parametri, ovviamente questi vengono trasmessi, ma non stampati. Alcuni codici vanno usati insieme ad altri, altrimenti perdono significato.
Tieni presente che i codici di controllo hanno in generale significato diverso se diretti al video invece che alla stampante; gli unici che producono lo stesso effetto sono 10, 13, 18 e 146.

CODICE	SIGNIFICATO	PARAMETRI	NUM. BYTE
8	MODO GRAFICO	no	1
10	invio LINE FEED e RETURN	no	1
13	invio LINE FEED e RETURN	no	1

14	MODO CARATTERE ALLARGATO	no	1
15	MODO CARATTERE NORMALE	no	1
16	SPOSTAMENTO PO-SIZIONE STAMPA A UNA COLONNA	2	3
17	PASSAGGIO SET MINUSC./MAIUSC.	no	1
18	MODO RVS=ON	no	1
26	MODO RIPETIZIO-NE CAR. GRAFICO	1	2
27	SPOSTAMENTO PO-SIZIONE GRAFICA deve essere se-guito da CHR$(16)	2	4
145	PASSAGGIO SET MAIUSC./GRAFICO	no	1
146	MODO RVS=OFF	no	1

Esaminiamo dettagliatamente i singoli codici di controllo, riportando alcuni programmi esempio. In alcuni programmi esempio abbiamo usato la tecnica di far lavorare il programma stampando i risultati, poi il programma lista se stesso; in conseguenza, in questi casi, vedrai prima i risultati e poi il listato del programma.

CHR$(8)   MODO GRAFICO

Predispone la stampa in modo grafico; tale modo resta attivo fino all'invio dei codici di controllo 14 o 15, che riportano rispettivamente in modo carattere allargato e in modo carattere normale.

Dopo aver attivato il modo grafico devi passare nella lista di stampa i codici dei caratteri grafici da stampare, come stringa. Ogni carattere grafico e' formato da una colonna di 7 punti, e viene codificato secondo le seguenti regole:

.i punti da disegnare devono corrispondere alla cifra 1, quelli da non disegnare devono corrispondere alla cifra 0,

.le 7 linee hanno pesi diversi, partendo dall'alto i pesi sono: 1, 2, 4, 8, 16, 32, 64, cioe' le potenze di 2, partendo dall'esponente 0 e arrivando all'esponente 6,

.devi moltiplicare ogni cifra per il peso corrispondente e sommare i valori, per una colonna con 7 cifre 1 ottieni 1+2+4+8+16+32+64=127, per una colonna con 7 cifre 0 ottieni 0,

.al numero ottenuto devi aggiungere 128, in tale modo il codice calcolato puo' variare da 128 a 255.

Il codice deve essere passato nella "lista" di stampa con la funzione CHR$. Se desideri ottenere un disegno composto da piu' colonne, devi ripetere il procedimento spiegato per ogni colonna.

Abbiamo preparato il disegno di un omino con le braccia alzate, occupando 11 colonne di 7 punti; lo riportiamo indicando con asterischi i punti da disegnare e con lineette quelli da lasciare in bianco. Inoltre riportiamo di fianco l'immagine ottenuta con 0 e 1 e con a margine i pesi di ogni punto e sotto ogni colonna il codice risultante dal calcolo.

```
-**-***-**-    1)  0  1  1   0  1  1  1   0  1  1  0
--**-*-**--    2)  0  0  1   1  0  1  0   1  1  0  0
---*****---    4)  0  0  0   1  1  1  1   1  0  0  0
----***----    8)  0  0  0   0  1  1  1   0  0  0  0
----***----   16)  0  0  0   0  1  1  1   0  0  0  0
---**-**---   32)  0  0  0   1  1  0  1   1  0  0  0
-***---***-   64)  0  1  1   1  0  0  0   1  1  1  0

Valore codici:     0 65 67 102 61 31 61 102 67 65  0
```

Aggiungendo 128 agli 11 codici otteniamo:
128, 193, 195, 230, 189, 159, 189, 230, 195, 193, 128,
passando dopo CHR$(8) le funzioni CHR$ degli 11 codici otteniamo il disegno di un omino. Il nostro disegno e' simmetrico e inizia e finisce con una colonna bianca; per questo disegnando vicino alcuni omini essi non risultano attaccati insieme.

Nel programma COD8-1 abbiamo inserito gli 11 numeri con una linea DATA, la 4; poi prepariamo nella stringa A$ l'omino completo come stringa; stampando A$, in modo grafico, otteniamo il disegno.

```
1 rem cod8-1
2 open10,4,7
3 cmd10
4 data0,65,67,102,61,31,61,102,67,65,0
5 a$="":forj=1to11
6 reada:a=a+128
7 a$=a$+chr$(a)
8 nextj
9 fork=1to5
10 printchr$(8)a$;
11 nextk:print:print
12 fork=1to7
13 printa$;
14 nextk:print:print
15 fork=1to9
16 printa$;
```

```
17 nextk:print:print
18 fork=1to11
19 printa$;
20 nextk:print
21 printchr$(15)
22 print:list:print#10:close10
```

COMMENTO A COD8-1

.2/3: apre la stampante con lfn=10 e sa=7, con CMD10 trasferisce l'uscita video alla stampante.

.4: linea DATA con i codici delle 11 colonne di punti, prima di aggiungere 128.

.5/8: legge gli 11 codici, aggiunge 128 a ogni codice e costruisce la stringa A$ con il disegno.

.9/11: stampa 5 omini, dopo aver attivato la stampa grafica con CHR$(8). Il codice e' usato all'interno del ciclo FOR, e quindi viene inviato 5 volte; bastava inviarlo una sola volta prima del ciclo FOR. Alla linea 11 esegue 2 volte PRINT, la prima volta per andare a capo e la seconda per produrre uno spazio. In modo grafico la distanza tra le righe e' minore rispetto al modo carattere.

.12/14: stampa 7 omini, lavorando come sopra, ma senza inviare di nuovo il codice 8.

.15/17: stampa 9 omini.

.18/20: stampa 11 omini.

.21: ritorna in modo carattere normale stampando CHR$(15)

.22: lista se stesso, chiude la linea e il file.

Nel programma esempio COD8-2, abbiamo disegnato una casetta che occupa un rettangolo di 17 colonne e 21 righe. Per poterla stampare la dobbiamo dividere a strisce alte 7 punti ciascuna; otteniamo 3 strisce. Stamperemo le tre parti una per riga, una sotto l'altra.
Per evitare la fatica di calcolare i codici abbiamo disegnato la casetta in 21 linee di programma, 21 linee DATA, scritte una dopo l'altra. Prima, con una REM abbiamo numerato le colonne, per disegnare piu' facilmente. La casetta viene disegnata usando gli asterischi.
Abbiamo incorporato nel programma una routine che legge le 21 stringhe, 7 per volta, analizza i caratteri delle stringhe e sostituisce 1 agli asterischi e 0 agli spazi. Poi calcola il valore di ogni colonna, aggiunge 128 e costruisce la stringa con la funzione CHR$ di ogni codice.
Puoi estrarre questa routine e adattarla a disegni di altre dimensioni. Ricordati di usare un numero di righe multiplo di 7.
Il programma stampa 10 volte la casetta e poi lista se stesso.

```
1 REM COD8-2
3 DIMT$(7),C$(3):REM STRINGHE PER DISEGNO
4 DIMT(6):REM PER CALCOLO
8 REM DISEGNO CASETTA IN 21 LINEE DATA
9 REM  12345678901234567
10 DATA"                 "
11 DATA"               * "
12 DATA"               * "
13 DATA"        *     *  "
14 DATA"       * *   *   "
15 DATA"      *   * ***  "
16 DATA"     *     ** *  "
17 DATA"    *       * *  "
18 DATA"   *         **  "
19 DATA"  *           *  "
20 DATA" *************** "
21 DATA" *             * "
22 DATA" * ***     *** * "
23 DATA" * * *     * * * "
24 DATA" * ***     *** * "
25 DATA" *             * "
26 DATA" *             * "
27 DATA" *     ***     * "
28 DATA" *     * *     * "
29 DATA" *     * *     * "
30 DATA" *************** "
36 PRINT"♥STO CALCOLANDO I CODICI DELLA CASETTA"
40 REM CALCOLO COLONNE DI PUNTI
45 FORK=1TO3:C$(K)="":FORJ=1TO7
50 READT$(J):PRINTT$(J):NEXTJ
55 FORL=1TO17:I=0
60 FORJ=1TO7:T(I)=ASC(MID$(T$(J),L,1))
65 IFT(I)=42THENT(I)=1:ELSET(I)=0
70 I=I+1:NEXTJ
75 T=0:FORJ=0TO6:T=T+T(J)*2↑J:NEXTJ:T=T+128
80 C$(K)=C$(K)+CHR$(T):NEXTL
85 NEXTK
100 REM STAMPA CASETTA
105 OPEN4,4:PRINT#4,CHR$(8)
110 FORJ=1TO3:FORK=1TO10
115 PRINT#4,C$(J);:NEXTK:PRINT#4
120 NEXTJ
125 PRINT#4,CHR$(15)
130 CMD4:LIST:PRINT#4:CLOSE4
```

COMMENTO A COD8-2

.3: dimensiona T$(7) per contenere le 7 stringhe di una striscia, e C$(3) per contenere le 3 stringhe da stampare per formare il disegno.

.4: dimensiona un vettore T(6) per contenere i 7 numeri corrispondenti a una colonna di punti.

.8/30: disegno della casetta in linee DATA.

.36: stampa un messaggio per avvisare che esegue il calcolo dei codici.

.40/85: ciclo per calcolare i codici della casetta.

.45: inizia il ciclo per K da 1 a 3 per calcolare le 3 stringhe che costruiscono il disegno, pulisce la stringa e inizia un ciclo per J da 1 a 7 per leggere 7 linee DATA.

.50: legge la stringa, la stampa sul video, disegnando cosi' la casetta con gli asterischi, e chiude il ciclo di J.

.55: inizia il ciclo per analizzare i 17 caratteri di ogni stringa, e pone I=0, indice per il vettore T.

.60/70: preleva da ognuna delle 7 stringhe i 7 caratteri che occupano la stessa posizione verticale e trasforma gli asterischi in 1 e gli spazi in 0 prima di porli nel vettore T(I). Alla fine del ciclo il vettore T(I) contiene una colonna di punti in cifre 1 e 0.

.75: calcola il codice corrispondente moltiplicando le cifre per i relativi pesi, poi aggiunge 128.

.80: aggiunge alla stringa C$(K) il nuovo carattere grafico calcolato e chiude il ciclo di L. Alla fine di questo ciclo la stringa C$(K) contiene tutti i 17 codici della striscia relativa, trasformati in carattere ASCII.

.85: chiude il ciclo di K. Alla fine sono pronte le 3 stringhe C$(K).

.100/120: stampa 10 casette ripetendo 10 volte ogni disegno e poi andando a capo.

.125: disattiva il modo grafico e torna al modo carattere normale.

.130: lista se stesso e chiude.

CHR$(10) e CHR$(13) INVIO LINE FEED E RETURN

Questi due codici hanno lo stesso comportamento, ognuno di essi stampa un LINE FEED e un RETURN e provoca la stampa di tutto quello che e' contenuto nel buffer. Se termini una "lista" con uno di questi codici, e non aggiungi il ";" finale, la mancanza di punteggiatura provoca un ulteriore RETURN.
Nel programma COD10/13 che segue, ti mostriamo come l'effetto dei due codici sia il medesimo e come influisce il valore di "lfn" sulla spaziatura tra le linee.

```
1 REM COD10/13
5 OPEN129,4
10 A$="PROVA1 LFN>128":B$="PROVA DI CHR$(10)"
15 PRINT#129,A$CHR$(10)B$
20 C$="PROVA2 LFN>128":D$="PROVA DI CHR$(13)"
25 PRINT#129,C$CHR$(13)D$
30 CLOSE129
35 OPEN10,4
40 E$="PROVA3 LFN<128":F$="PROVA DI CHR$(10)"
45 PRINT#10,E$CHR$(10)F$
50 G$="PROVA4 LFN<128":H$="PROVA DI CHR$(13)"
55 PRINT#10,G$CHR$(13)H$
60 CLOSE10
65 STOP
```

RISULTATI PROGRAMMA COD10/13

```
PROVA1 LFN>128
PROVA DI CHR$(10)

PROVA2 LFN>128
PROVA DI CHR$(13)

PROVA3 LFN<128
PROVA DI CHR$(10)
PROVA4 LFN<128
PROVA DI CHR$(13)
```

COMMENTO A COD10/13
.5: apre con lfn=129 (>127) e questo provoca una doppia spaziatura a fine linea se manca la punteggiatura finale.
.10/15: prepara le due stringhe A$ e B$ e le stampa separandole con CHR$(10); esse vengono stampate una su ogni riga, ma alla fine si ha un doppio spazio.
.20/25: come sopra, ma separando le due stringhe con CHR$(13); si ottiene lo stesso effetto di prima.
.30: chiude il file con lfn=129.
.35: apre il file con lfn=10 (<127) e questo provoca una spaziatura semplice in assenza di punteggiatura finale.
.40/60: stampa con gli stessi codici di controllo e ottiene la spaziatura semplice anche in assenza di punteggiatura finale.
Se la punteggiatura finale invece di essere un ";" e' una ",", si ha l'aggiunta di 10 spazi, e questo puo' provocare il passaggio a nuova linea, anche se non sono presenti o il codice 10 o il codice 13.

CHR$(14) MODO CARATTERE ALLARGATO
Predispone la stampa in modo testo del carattere allargato, cioe' del carattere formato da 12 punti per riga e 7 punti per colonna. La predisposizione rimane fino a quando si invia il codice 15, per tornare al carattere normale, o il codice 8 per passare in modo grafico.
Segue il programma COD14-1, che stampa due linee in carattere allargato, poi torna al modo normale, lista se stesso e chiude correttamente la comunicazione.

```
1 REM COD14-1
2 OPEN10,4
3 CMD10
4 PRINTCHR$(14)"COMMODORE"
5 PRINTCHR$(14)" MPS-803 "
6 PRINTCHR$(15)
7 LIST
8 PRINT#10:CLOSE10
```

Il programma COD14-2, invece, mostra come si possono ottenere sulle stesse linee di stampa sia caratteri normali che caratteri allargati, usando alternativamente i codici 14 e 15.

```
1 REM COD14-2
2 OPEN10,4
3 PRINT#10,CHR$(14)"COMMODORE ";
4 PRINT#10,CHR$(15)"COMMODORE "
5 PRINT#10,CHR$(14)" MPS-803 ";
6 PRINT#10,CHR$(15)" MPS-803 "
7 FORK=65TO68
8 PRINT#10,CHR$(14)CHR$(K)CHR$(15)CHR$(K);
9 NEXTK
10 PRINT#10,CHR$(15):CLOSE10
```

In questo caso stampiamo con PRINT#10 e non trasferendo la stampa dal video alla stampante con CMD10. Nota alle linee 7/9, come otteniamo in ciclo la stampa alternata di un carattere allargato e uno normale; inoltre scriviamo le variabili striga una vicino all'altra senza punteggiatura, ma dobbiamo porre un ";" finale per evitare che vada a capo.

RISULTATI PROGRAMMA COD14-2

```
COMMODORE  COMMODORE
 MPS-803   MPS-803
AABBCCDD
```

CHR$(15) MODO CARATTERE NORMALE
Predispone la stampa nel modo normale, che e' attivo all'accensione. Devi usare questo codice per disattivare sia il modo a carattere allargato, che il modo grafico.

CHR$(16) SPOSTAMENTO POSIZIONE STAMPA A UNA COLONNA
Questo codice predispone l'inizio della stampa a una colonna, tra 00 e 79. Il numero della colonna, espresso come stringa di 2 caratteri, deve seguire immediatamente CHR$(16). Ricorda che le cifre numeriche hanno codice ASCII che varia da 48 a 57; per indicare la colonna 18 puoi scrivere "18" oppure CHR$(49)CHR$(56).
Il codice 16 puo' essere usato sia in modo testo che in modo grafico. Riportiamo un esempio nel programma COD16-1.

```
01234567890123456789012345678901234567890123456789
 0 X            1 X            2 X
01234567890123456789012345678901234567890123456789
X 0            X 1            X 2
```

```
1 REM COD16-1
5 OPEN10,4
```

```
10 CMD10:A$=""
15 DATA0,65,67,102,61,31,61,102,67,65,0
20 DATA48,48,49,53,51,48
25 FORI=1TO11:READA
30 A$=A$+CHR$(A+128)
35 NEXTI
40 GOSUB100
50 FORI=0TO2:READX,Y
55 PRINTCHR$(15)CHR$(16)CHR$(X)CHR$(Y);I;CHR$(8)A$;
60 NEXTI:PRINTCHR$(15)
65 GOSUB100:RESTORE20:FORI=0TO2:READX,Y
70 PRINTCHR$(8)CHR$(16)CHR$(X)CHR$(Y);A$;CHR$(15);I;
75 NEXTI
80 PRINTCHR$(15):LIST
85 PRINT#10:CLOSE10:STOP
100 FORK=0TO4:FORI=0TO9:PRINTCHR$(48+I);
105 NEXTI:NEXTK:PRINT:RETURN
```

COMMENTO A COD16-1

.5/10: apre il file logico 10 per la stampante, trasferisce l'uscita video alla stampante e pulisce la stringa A$.

.15: linea DATA che contiene i codici dell'omino con le braccia alzate, precedentemente preparato; esso occupa 11 colonne di punti.

.20: linea DATA che contiene 3 coppie di cifre, in codice ASCII, per definire le tre colonne: 00, 15, 30.

.25/35: preparazione in A$ del disegno dell'omino, aggiungendo 128 a ogni codice e trasformandolo in stringa.

.40: esecuzione del sottoprogramma in 100 per stampare una linea di numerazione delle posizioni di stampa.

.50/60: stampa in ciclo, dopo aver letto le cifre di ogni colonna dalla linea DATA 20, aver definito la posizione di stampa, del numero I e dell'omino. Nota nei risultati che le colonne selezionate sono 00, 15 e 30, che i numeri sono stampati preceduti e seguiti da uno spazio, e che gli omini occupano quasi due posizioni carattere. In questo caso il codice 16 viene usato trovandosi in modo carattere per effetto del codice 15. Concluso il ciclo, va a capo e ripristina il modo carattere con il codice 15.

.65/75: esegue nuovamente il sottoprogramma per numerare le posizioni di stampa, poi passa con il codice 8 in modo grafico e usa il codice 16 per definire la posizione della colonna di inizio stampa. Nel ciclo viene stampato prima l'omino e poi il numero I, dopo essere tornati in modo carattere con il codice 15. Nota nei risultati che: gli omini iniziano esattamente alle colonne 00, 15 e 30 (la numerazione va da 0 a 79), mentre i numeri iniziano esattamente 8 punti dopo l'omino, cioe' risultano sfalsati rispetto alla sovrastante colonna di numerazione.

.80/85: ripristino del modo carattere, lista del programma e chiusura del file.

.100/105: sottoprogramma di numerazione linea di stampa da 0 a 9 per 5 volte.

CHR$(17) PASSAGGIO AL SET MINUSCOLO/MAIUSCOLO

Questo codice fa passare al set minuscolo/maiuscolo, con validita' locale, cioe' solo per l'istruzione di PRINT in corso (fino al primo RETURN), indipendentemente dal set selezionato con l'istruzione OPEN, che torna attivo al termine della PRINT.

CHR$(18) e CHR$(146) MODI RVS-ON E RVS-OFF

Il codice 18 predispone la stampa in campo inverso: RVS-ON, mentre il codice 146 predispone la stampa normale: RVS-OFF. Il primo ha validita' locale, cioe' resta valido fino al primo carattere RETURN. Per questa ragione, se non si desidera alternare sulla stessa linea i due modi di stampa, non e' necessario usare il codice 146.
Il programma COD18/146-1, che segue, illustra quanto detto.

```
10 CMD10:A$=""
CARATTERE NORMALE
CARATTERE IN CAMPO INVERSO
CAMBIOCAMBIOCAMBIOCAMBIO

1 REM COD18/146-1
5 A$="CARATTERE NORMALE"
10 B$="CARATTERE IN CAMPO INVERSO"
15 C$="CAMBIO"
20 OPEN4,4:CMD4
25 PRINTA$
30 PRINTCHR$(18)B$CHR$(146)
35 FORK=1TO2
40 PRINTCHR$(18)C$CHR$(146)C$;
45 NEXTK:PRINT
50 LIST:PRINT#4
55 CLOSE4:STOP
```

Il modo RVS-ON puo' essere usato solo per la stampa in modo testo, carattere normale o allargato.
Questi due codici possono essere usati con lo stesso effetto per la stampa sul video.

CHR$(26) RIPETIZIONE CARATTERE GRAFICO

Questo codice di controllo deve essere usato dopo essere entrati in modo grafico con il codice 8. Esso deve essere seguito da un numero N, che specifica quante volte deve essere ripetuta la colonna di punti che segue; tale numero puo' variare da 0 a 255 e deve essere passato con la funzione CHR$(N). Se N=0 il carattere viene ripetuto 256 volte. Se vuoi ripetere per un numero maggiore di volte devi usare la sequenza CHR$(26)CHR$(N)...; piu' volte. Dopo CHR$(N) deve comparire il codice che rappresenta la colonna di punti da ripetere, passato con la funzione CHR$.
Il programma COD26-1 esemplifica l'uso del codice 26 per allargare il nostro omino, gia' usato in qualche esempio, agendo su ogni colonna di punti che lo compone.

```
1 REM COD26-1
5 OPEN10,4
10 CMD10
15 DATA0,65,67,102,61,31,61,102,67,65,0
20 FORK=1TO5
25 FORI=1TO11
30 READA:A$=CHR$(A+128)
35 PRINTCHR$(8)CHR$(26)CHR$(2)A$;
40 NEXTI
45 RESTORE:NEXTK
50 PRINT:PRINT
55 FORK=1TO5
60 FORI=1TO11
65 READA:A$=CHR$(A+128)
70 PRINTCHR$(8)CHR$(26)CHR$(3)A$;
75 NEXTI
80 RESTORE:NEXTK
85 PRINT:PRINTCHR$(15)
90 PRINT#10:CLOSE10
```

RISULTATI COD26-1

COMMENTO A COD26-1

.5/10: apre la stampante con lfn=10 e trasferisce l'uscita video al file logico 10.

.15: linea DATA che contiene la codifica dell'omino, senza l'aggiunta di 128 ad ogni codice.

.20/45: ripete ciclicamente 5 volte la stampa dell'omino. Legge un codice per volta, aggiunge 128, trasforma in stringa in A$, stampa 2 volte ogni colonna di punti con la sequenza CHR$(8)CHR$(26)CHR$(2)A$. Il codice 8 poteva essere usato una sola volta fuori dal ciclo. RESTORE rende di nuovo attiva la linea DATA.

.50/75: esegue di nuovo la stampa allargata dell'omino ripetendo 3 volte ogni colonna di punti.

.85/90: ripristina il modo carattere e chiude correttamente.

Il codice 26 e' molto utile per stampare istogrammi orizzontali. Nel programma COD26-2 abbiamo realizzato un esempio.

```
1 REM COD26-2
3 SA=7
5 RESTORE:OPEN10,4,SA
10 CMD10:G$=CHR$(252)
15 DATA18,20,22,38,50,48,55,49,70,39,45,65
20 PRINTCHR$(14)"ANDAMENTO VENDITE"
23 PRINT"ANNI 1973/1984"
25 PRINT
```

```
30 FORI=1TO12
35 READB
40 C=1972+I
45 PRINTCHR$(15)C;"      ";CHR$(8)CHR$(26)CHR$(B)G$
50 NEXTI
51 PRINT:PRINT
53 IFSA=7THENSA=0:PRINT#10:CLOSE10:GOTO5
55 PRINT#10,CHR$(15):CLOSE10:STOP
```

RISULTATI COD26¬2

COMMENTO A COD26¬2

.3: pone SA=7, per attivare il set minuscolo/maiuscolo.

.5/10: esegue il RESTORE e apre la stampante. Trasferisce l'uscita video alla stampante e definisce il carattere grafico G$, che corrisponde a una colonna di punti con spenti i due punti piu' in alto; cosi' le barrette dell'istogramma non si toccano.

.15: linea DATA con 12 numeri che rappresentano l'andamento delle vendite in 12 anni.

.20/25: stampa l'intestazione della tabella.

.30/51: stampa le 12 linee della tabella. Per ogni linea stampa in modo carattere l'anno e in modo grafico la barretta ripetendo il carattere G$ tante volte quanto e' il valore B. Nota che va a capo in modo grafico, e quindi i numeri degli anni risultano un po' ravvicinati verticalmente.

.53: se SA=7 pone SA=0 e ritorna alla linea 5 dopo aver chiuso il file. Cosi' stampa una seconda volta la tabella, ma con l'intestazione nell'altro set di caratteri.
.55: se SA=0 ritorna in modo carattere e chiude correttamente.

CHR$(27) SPOSTAMENTO POSIZIONE GRAFICA
Consente di spostare la posizione di inizio della stampa in uno dei 480 punti di una linea, da 0 a 479. Esso puo' essere usato sia in modo testo che in modo grafico. Non puo' pero' essere usato da solo, ad esso deve seguire il codice 16, seguito a sua volta dalla posizione del punto espressa in due byte, HI e LO, passati con la funzione CHR$. Per esempio se vuoi stampare a partire dal punto 323, devi eseguire il seguente calcolo:

```
X=INT(323/256)
Y=323-X*256
```

e usare nell'istruzione PRINT la sequenza:

```
CHR$(27)CHR$(16)CHR$(X)CHR$(Y)...
```

Nel programma COD27-1 stampiamo un gruppo di omini sovrapposti iniziando la prima volta nel punto 33, la seconda nel punto 22, la terza nel punto 11 e la quarta nel punto 0. I due byte HI e LO che danno la posizioni di inizio devono essere sempre passati, anche se il byte HI e' nullo.

```
1 REM COD27-1
5 OPEN10,4,7:CMD10
10 DATA0,65,67,102,61,31,61,102,67,65,0
15 A$(I)="":FORJ=1TO11:READA:A=A+128
20 A$=A$+CHR$(A):NEXTJ
25 FORK=0TO4
30 PRINTCHR$(8)CHR$(27)CHR$(16)CHR$(0);
35 PRINTCHR$(11*(3+K))A$;
40 NEXTK:PRINT
45 FORK=0TO6
50 PRINTCHR$(8)CHR$(27)CHR$(16)CHR$(0);
55 PRINTCHR$(11*(2+K))A$;
60 NEXTK:PRINT
65 FORK=0TO8
70 PRINTCHR$(8)CHR$(27)CHR$(16)CHR$(0);
75 PRINTCHR$(11*(1+K))A$;
80 NEXTK:PRINT
85 FORK=0TO10
90 PRINTCHR$(8)A$;
95 NEXTK:PRINT
100 PRINT:PRINTCHR$(15)
105 PRINT#10:CLOSE10
```

RISULTATI COD27-1

COMMENTO A COD27¬1

.5: apre il file della stampante con lfn=10 e trasferisce l'uscita video ad esso.

.10: linea DATA che definisce l'omino.

.15/20: costruzione in A$ dell'omino.

.25/40: stampa ciclica di 5 omini a partire dalle colonne: 33, 44, 55, 66, 77.

.45/60: stampa ciclica di 7 omini a partire dalle colonne: 22, 33, 44, 55, 66, 77, 88.

.65/80: stampa ciclica di 9 omini a partire dalle colonne: 11, 22, 33, 44, 55, 66, 77, 88, 99.

.85/95: stampa ciclica di 11 omini a partire dalla colonna 0.

.100/105: ritorno alla stampa in modo carattere e chiusura corretta.

Nella stampa delle prime 3 linee bastava posizionarsi al primo punto, dopo essere passati in grafica, fuori ciclo e poi proseguire la stampa in ciclo senza ulteriori posizionamenti.

Nel programma COD27¬2 stampiamo 7 tacche nelle posizioni punto 0, 50, 100, 150, 200, 250 e 300. Abbiamo stampato una linea di numeri per rendere riconoscibili le posizioni delle tacche.

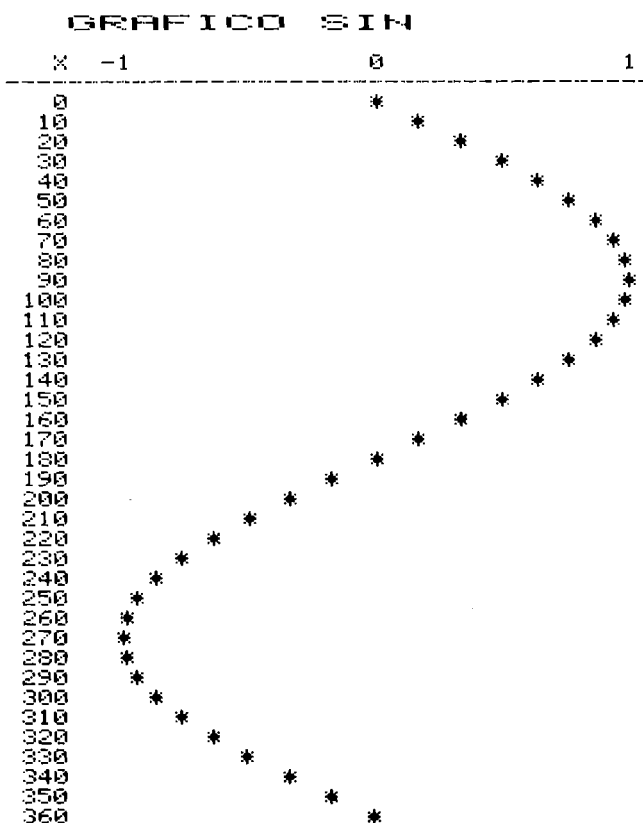
```
1 REM COD27-2
3 OPEN10,4
5 CMD10
7 FORK=0TO4:FORI=0TO9:PRINTCHR$(48+I);
9 NEXTI:NEXTK:PRINT
11 FORI=0TO6
13 A=50*I
15 B=INT(A/256)
17 C=A-B*256
19 PRINTCHR$(8)CHR$(27)CHR$(16);
21 PRINTCHR$(B)CHR$(C)CHR$(255);
23 NEXTI:PRINT#10,CHR$(15):CLOSE10
```

RISULTATI COD27¬2

```
01234567890123456789012345678901234567890123456789
|        |        |        |        |        |        |
```

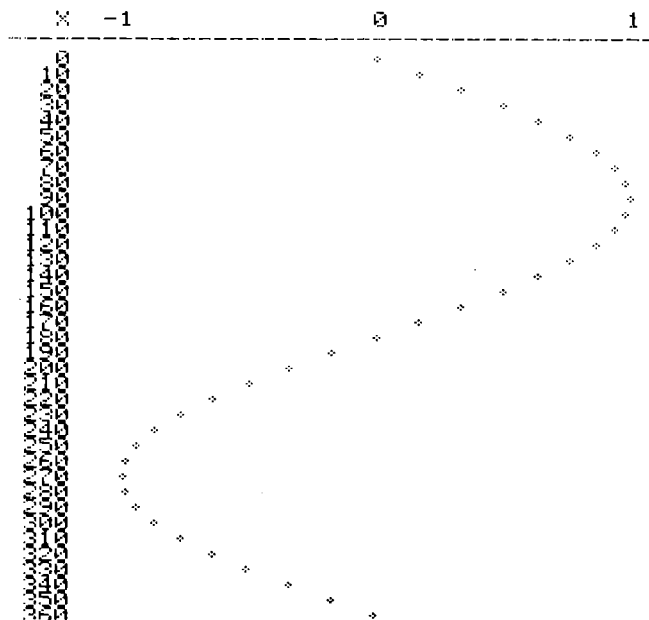
Nota come viene calcolata la posizione punto alle linee 13/17.

Come ultimi esempi dell'uso dei codici 27 e 16 riportiamo i programmi GRAFICO1 e GRAFICO2.

```
1  REM GRAFICO1
3  OPEN4,4:CMD4
5  D$=CHR$(14):N$=CHR$(15)
7  P$=CHR$(16):G$=CHR$(27)
9  C=23:A=16:O=4
11 A$="-":FORI=0TOC+A:A$=A$+"-":NEXT
13 S$="          "
15 PRINTD$"  GRAFICO SIN"
17 PRINTN$
19 PRINTLEFT$(S$,O-1)+"X";
21 PRINTSPC(C-A-O-1)"-1";
23 PRINTSPC(A-1)"0";
25 PRINTSPC(A-1)"1"
27 PRINTA$
29 FORI=0TO360STEP10
31 I$=RIGHT$(S$+STR$(I),O)
33 YO=C*6+A*6*SIN(I*π/180)
35 YH=INT(YO/256):YL=YO-YH*256
37 PRINTI$G$P$CHR$(YH)CHR$(YL)"*"
39 NEXTI:PRINT#4,N$:CLOSE4:STOP
```

COMMENTO A GRAFICO1

.3: apre la stampante e trasferisce ad essa l'uscita video.
.5/7: definisce come stringhe i caratteri di controllo 14, 15, 16 e 27.
.9: inizializza alcune costanti.
.11/13: prepara le stringhe A$ e S$.
.15: stampa l'intestazione in caratteri allargati.
.17/25: stampa X, ¬1, 0, 1 usando la funzione SPC per posizionarsi.
.27: stampa le lineette.
.29/39: ciclo di stampa della funzione SIN tra 0 e 360, calcolando le coordinate dei punti e stampandoli con il carattere asterisco in modo testo. Il posizionamento avviene con CHR$(27)CHR$(16) seguiti dalla posizione punto calcolata alle linee 33/35.
Questo grafico e' ottenuto lavorando in modo testo.

Il programma GRAFICO2, invece, lavora in modo grafico, stampando un piccolo rombo definito in F$ con 3 colonne di punti. Come puoi vedere dai risultati il grafico risulta molto piu' compatto.

```
1 REM GRAFICO2
3 OPEN4,4:CMD4
5 D$=CHR$(14):N$=CHR$(15):GR$=CHR$(8)
7 P$=CHR$(16):G$=CHR$(27):NO$=CHR$(15)
9 C=23:A=16:O=4:F$=CHR$(136)+CHR$(148)+CHR$(136)
11 A$="-":FORI=0TOC+A:A$=A$+"-":NEXT
13 S$="              "
15 PRINTD$"  GRAFICO SIN"
17 PRINTN$
19 PRINTLEFT$(S$,O-1)+"X";
21 PRINTSPC(C-A-O-1)"-1";
23 PRINTSPC(A-1)"0";
25 PRINTSPC(A-1)"1"
27 PRINTA$
29 FORI=0TO360STEP10
31 I$=RIGHT$(S$+STR$(I),O)
33 YO=C*6+A*6*SIN(I*π/180)
35 YH=INT(YO/256):YL=YO-YH*256
37 PRINTNO$I$GR$G$P$CHR$(YH)CHR$(YL)F$
39 NEXTI:PRINT#4,N$:CLOSE4:STOP
```

CHR$(145) PASSAGGIO SET MAIUSCOLO/GRAFICO
Questo codice fa passare al set maiuscolo/grafico in modo temporaneo, cioe' con validita' limitata, fino al primo carattere RETURN. Rimane preponderante la definizione del set di caratteri operata con il valore di "sa" al momento della OPEN.

5.4 GESTIONE DEL BUFFER DI STAMPA

Il buffer della MPS-803 puo' contenere 90 caratteri; si ha la stampa automatica quando il buffer e' pieno. La stampa puo' aver luogo producendo lo svuotamento totale o parziale del buffer. Nel buffer vengono memorizzati tutti i caratteri che il calcolatore invia, quelli stampabili, i caratteri separatori, le funzioni di stampa, i caratteri di controllo.
Quando usi un programma che manda dati alla stampante, non sempre vedrai uscire dati in corrispondenza all'esecuzione di istruzioni PRINT; se non operi bene puoi chiudere il file di stampa con CLOSE senza aver svuotato completamente il buffer (questo non

succede se prima della CLOSE usi un'istruzione PRINT senza lista dati).
Riassumiamo le condizioni nelle quali avviene la stampa:

.1) Il buffer e' pieno, cioe' contiene 90 caratteri, ma i caratteri di testo stampabili sono meno di 80, o tra caratteri di testo e caratteri grafici sono presenti meno di 480 punti. Il buffer viene svuotato completamente producendo una linea di stampa senza andare a capo (infatti non e' stato incontrato un carattere di codice 10 o 13 che manda a capo).

.2) Il buffer contiene meno di 90 caratteri, ma riceve un carattere di "vai a capo"; si ha la stampa di tutti i caratteri con svuotamento completo del buffer e si va a capo.

.3) Il buffer non e' pieno, ma contiene tra caratteri di testo e grafici piu' di 480 punti stampabili; si ha la stampa di una linea con vai a capo, e il buffer viene svuotato dei caratteri stampati.

Il programma ST-AUTOM esemplifica quanto detto.

```
1 REM ST-AUTOM
2 M0$="STAMPA 80 CARATTERI, CHR$(13) FINALE"
3 M1$="STAMPA 90 CARATTERI, CHR$(13) FINALE"
4 M2$="STAMPA 90 CARATTERI, CON ; FINALE"
5 M3$="STAMPA 8 VOLTE CHR$(10); E POI 70"
6 M3$=M3$+" CARATTERI, CON CHR$(13); FINALE"
7 LF$=CHR$(10):CR$=CHR$(13)
8 A$="0123456789":B$=""
9 A1$="ABCDEFGHIJKLMNOPQRSTUVWXYZ":B1$=""
10 FORK=1TO8:B$=B$+A$:NEXTK
11 FORK=1TO4:B1$=B1$+A1$:NEXTK
12 M4$="STAMPA 12 VOLTE CHR$(13); POI 26 "
13 M5$=LEFT$(M4$,26)
14 M4$=M4$+"CARATTERI CON CHR$(13) FINALE"
15 M5$=M5$+"POI LE 10 CIFRE CON ; FINALE"
101 OPEN4,4
103 PRINT#4,M0$
105 PRINT#4,B$:GOSUB200
107 PRINT#4,M1$
109 PRINT#4,B$;A$:GOSUB200
111 PRINT#4,M2$
113 PRINT#4,B$;A$;:GOSUB200
115 PRINT#4,M3$
117 PRINT#4,LF$;LF$;LF$;LF$;LF$;LF$;LF$;
119 PRINT#4,LF$;LEFT$(B$,70);CR$;
121 PRINT#4,M4$
123 PRINT#4,CR$;CR$;CR$;CR$;CR$;CR$;CR$;
125 PRINT#4,CR$;CR$;CR$;CR$;CR$;A1$:GOSUB200
127 PRINT#4,M3$
129 PRINT#4,LF$;LF$;LF$;LF$;LF$;LF$;LF$;
131 PRINT#4,LF$;LEFT$(B1$,70);CR$;
133 PRINT#4,M5$
135 PRINT#4,"LE 10 CIFRE RESTANO NEL BUFFER"
137 PRINT#4,CR$;CR$;CR$;CR$;CR$;CR$;CR$;
139 PRINT#4,CR$;CR$;CR$;CR$;CR$;A$;:GOSUB200
141 CLOSE4:STOP
200 FORI=1TO1000:NEXTI:RETURN
```

Non riportiamo i risultati del programma, che potrai ottenere tu eseguendolo. Esso prepara diverse stringhe di diversa lunghezza e le stampa; dopo ogni stampa viene chiamata una routine che provoca un ciclo di attesa e che ti permette di vedere cosa e' stato stampato fino a quel momento. Ricorda che se la lista di stampa termina senza punteggiatura il sistema aggiunge un CHR$(13) e quindi si ha la stampa con a capo. Se la lista di stampa termina con ";" non viene aggiunto alcun carattere, mentre se essa termina con "," si ha l'aggiunta di 10 spazi.
Nel programma ST-AUTOM, dato che l'ultima lista di stampa termina con ";" (linea 139) e alla linea 141 si ha la CLOSE del file, gli ultimi caratteri restano nel buffer e non sono stampati. Prova alla fine del programma ad eseguire in immediato:

```
OPEN4,4:PRINT#4:CLOSE4
```

e li vedrai uscire.

5.5 COME SI COMPONE UNO STAMPATO

Nella preparazione degli stampati si presentano due problemi: gli allineamenti orizzontali e le spaziature verticali. Per quanto riguarda il primo problema abbiamo gia' visto in alcuni esempi che si puo' agire sulla posizione di stampa con:

- .le funzioni TAB e SPC,
- .la punteggiatura "," e ";",
- .i codici di controllo 16 e 27 seguiti da opportuni parametri,
- .le funzioni LEN, LEFT$, MID$ e RIGHT$, che consentono di preparare dati stringa tutti della stessa lunghezza.

In generale si allineano i numeri a destra o al punto decimale, e le parole a sinistra.
Per quanto riguarda il problema delle spaziature verticali, dato che la nostra stampante non contiene un contarighe automatico (con relativi codici di controllo per usufruirne), dobbiamo creare noi una routine che conti le righe di avanzamento della carta e ci consenta di intervenire per andare a nuovo foglio o per posizionarci a una determinata riga.
E' necessario conoscere il numero di righe del modulo che si usa e stabilire di quante righe deve essere il margine non stampato.
Il sottoprogramma CONTARIGHE ci consente di stampare su un foglio lungo 66 righe, con un margine di 6 righe. Se scriviamo GOSUB7000 otteniamo di contare una riga di stampa, andando a nuovo foglio se ne sono gia' state scritte 60. Se scriviamo GOSUB7040 otteniamo di andare a nuovo foglio comunque.

```
7000 REM CONTARIGHE
7005 IFCR<>0THEN7020
```

```
7010 NR=66:REM LUNGHEZZA FOGLIO
7015 RM=6:REM RIGHE DI MARGINE
7020 IFCR=NR-RMTHEN7030:REM CAMBIO FOGLIO
7025 CR=CR+1:RETURN:REM +1 IN CONTA RIGHE
7030 FORR=1TORM:PRINT#4:NEXTR:REM CAMBIA FOGLIO
7035 CR=1:RETURN:REM RICOMINCIA FOGLIO
7040 REM ENTRATA PER CAMBIARE FOGLIO
7045 FORR=CR+1TOMR+NR:PRINT#4:NEXTR:GOTO7035
```

Quando si chiama la prima volta il sottoprogramma con GOSUB7000 deve essere CR=0, e allora viene inizializzato NR=66 e RM=6 (puoi cambiare queste costanti secondo le tue esigenze). Quando, invece CR<>0, la routine va a nuovo foglio se necessario e incrementa il contatore di riga. L'entrata 7040 fa andare a nuovo foglio indipendentemente dal numero di righe gia' stampate.

Seguono alcuni esempi relativi agli allineamenti orizzontali; essi stampano i risultati e poi listano se stessi (puoi evitare la lista cancellando LIST).

Il programma ES5.1 esemplifica l'uso della funzione TAB, dopo aver trasferito la stampa dal video alla stampante con CMD.

```
0123456789012345678901234567890123456789
POS0      POS10               POS30

NUOVA RIGA

1 REM ES5.1
3 OPEN4,4:CMD4
5 A$="0123456789":B$=""
7 FORK=1TO4:B$=B$+A$:NEXTK
9 PRINTB$
11 PRINTTAB(0)"POS0";TAB(6)"POS10";
13 PRINTTAB(15)"POS30"
15 PRINTTAB(80);"NUOVA RIGA"
17 PRINT:LIST
19 PRINT#4:CLOSE4:STOP
```

Il programma ES5.2 esemplifica l'uso della funzione SPC, dopo aver trasferito la stampa dal video alla stampante con CMD.

```
0123456789012345678901234567890123456789
DIECI CAR.          DIECI CAR.
DIECI CAR.          DIECI CAR.

1 REM ES5.2
3 OPEN4,4:CMD4
5 A$="0123456789":B$=""
7 FORK=1TO4:B$=B$+A$:NEXTK
9 PRINTB$
```

```
11 C$="DIECI CAR."
13 PRINTC$;SPC(10);C$
15 PRINTC$TAB(10)C$
17 PRINT:LIST
19 PRINT#4:CLOSE4:STOP
```

Puoi modificare ES5.1 e ES5.2 per stampare con PRINT#, invece che con CMD; ottieni gli stessi risultati.

Il programma ES5.3 esemplifica l'uso della punteggiatura finale nella lista di stampa. Esso stampa con CMD.

```
0123456789012345678901234567890123456789
ABC          DEF
PRIMA PAROLA          SECONDA PAROLA
ABCDEF
PRIMA PAROLASECONDA PAROLA
```

```
1 REM ES5.3
3 OPEN4,4:CMD4
5 A$="0123456789":B$=""
7 FORK=1TO4:B$=B$+A$:NEXTK
9 PRINTB$
11 PRINT"ABC","DEF"
13 PRINT"PRIMA PAROLA","SECONDA PAROLA"
15 PRINT"ABC";"DEF"
17 PRINT"PRIMA PAROLA";"SECONDA PAROLA"
19 PRINT:LIST
21 PRINT#4:CLOSE4:STOP
```

Puoi modificare ES5.3, stampando con PRINT#, invece che con CMD e ottenendo gli stessi risultati.

Il programma ES5.4 mostra come incolonnare i numeri riducendoli tutti della stessa lunghezza, dopo averli trasformati in stringa, con l'aggiunta di spazi a sinistra, usando la funzione RIGHT$.

```
0123456789012345678901234567890123456789
 1234    13567    45    890
 5432    9876    3456    12345

TABELLA NUMERI IN COLONNA

0123456789012345678901234567890123456789
      1234     13567        45       890
      5432      9876      3456     12345
```

```
1 REM ES5.4
5 DIMC(7),C$(7):SP$="         "
10 OPEN4,4:CMD4
15 A$="0123456789":B$=""
20 FORK=1TO4:B$=B$+A$:NEXTK
25 PRINTB$
30 DATA1234,13567,45,890
```

```
35 DATA5432,9876,3456,12345
40 FORK=0TO7:READC(K)
45 C$(K)=STR$(C(K)):NEXTK
50 FORJ=0TO1:FORK=0TO3:PRINTC(K+J*4);"   ";
55 NEXTK:PRINT:NEXTJ
60 T$="TABELLA NUMERI IN COLONNA"
65 PRINT:PRINTT$:PRINT
70 PRINTB$
75 FORJ=0TO1:FORK=0TO3
80 PRINTRIGHT$(SP$+C$(K+J*4),10);
85 NEXTK:PRINT:NEXTJ
90 PRINT#4:CMD4:LIST:PRINT#4:CLOSE4:STOP
```

Il programma ES5.5 mostra come incolonnare i numeri ricorrendo, dopo la trasformazione in stringa, alle funzioni SPC e LEN.

```
0123456789012345678901234567890123456789
 1234     8976     3456     13567
 45     890     5432     9876

TABELLA NUMERI IN COLONNA

0123456789012345678901234567890123456789
      1234      8976      3456     13567
        45       890      5432      9876
```

```
1 REM ES5.5
5 DIMC(7),C$(7):SP$="          "
10 OPEN4,4:CMD4
15 A$="0123456789":B$=""
20 FORK=1TO4:B$=B$+A$:NEXTK
25 PRINTB$
30 DATA1234,8976,3456,13567,45,890,5432,9876
35 FORK=0TO7:READC(K):C$(K)=STR$(C(K)):NEXTK
40 FORJ=0TO1:FORK=0TO3:PRINTC(K+J*4);"   ";
45 NEXTK:PRINT:NEXTJ
50 T$="TABELLA NUMERI IN COLONNA"
55 PRINT:PRINTT$:PRINT:PRINTB$
60 FORJ=0TO1:FORK=0TO3
65 PRINTSPC(10-LEN(C$(K+J*4)))C$(K+J*4);
70 NEXTK:PRINT:NEXTJ
75 PRINT#4:CMD4:LIST:PRINT#4:CLOSE4:STOP
```

Il programma ES5.6 mostra come allineare parole ricorrendo alle funzioni SPC e LEN.

```
BELLO  BENISSIMO  ALLEGRO  PIACEVOLE
RILASSANTE  SOLEGGIATO  GRADEVOLE  UTILE
DEFINITIVO  ARIOSO  STUPENDO  MAGNIFICO

BELLO          BENISSIMO      ALLEGRO        PIACEVOLE
RILASSANTE     SOLEGGIATO     GRADEVOLE      UTILE
DEFINITIVO     ARIOSO         STUPENDO       MAGNIFICO
```

```
1 REM ES5.6
5 DIMC$(11)
10 OPEN4,4:CMD4
15 DATABELLO,BENISSIMO,ALLEGRO
20 DATAPIACEVOLE,RILASSANTE,SOLEGGIATO
25 DATAGRADEVOLE,UTILE,DEFINITIVO
30 DATAARIOSO,STUPENDO,MAGNIFICO
35 FORK=0TO11:READC$(K):NEXTK
40 FORJ=0TO2:FORK=0TO3:PRINTC$(K+J*4);"   ";
45 NEXTK:PRINT:NEXTJ
50 N=0:FORK=0TO11
55 IFLEN(C$(K))>NTHENN=LEN(C$(K))
60 NEXTK:PRINT:PRINT
65 N=N+3:FORJ=0TO2:FORK=0TO3
70 PRINTC$(K+J*4);SPC(N-LEN(C$(K+J*4)));
75 NEXTK:PRINT:NEXTJ
80 PRINT#4:CMD4:LIST:PRINT#4:CLOSE4:STOP
```

5.6 COPIA DEL VIDEO TESTO SU CARTA

In generale con il termine "copia del video su carta" (hardcopy) si intende uno stampato che corrisponda al contenuto del video. Tale stampato puo', nel nostro caso:

.Contenere gli stessi caratteri che compaiono sul video, grafici o di testo, ma non essere identico al video. Infatti sul video i caratteri sono rappresentati in una matrice 8x8, mentre sulla carta sono rappresentati in una matrice 6x7.

.Contenere una immagine identica ottenuta riproducendo esattamente i punti del video (con dimensioni diverse naturalmente).

Noi in questo paragrafo ci occupiamo della copia del video su carta, quando il video e' in modo testo; nel prossimo paragrafo trattiamo invece l'argomento del video grafico.
In modo testo la mappa del video occupa 1000 byte; l'indirizzo del primo byte si puo' ottenere moltiplicando per 256 il contenuto del byte 1342. Al momento dell'accensione la mappa video inizia al byte 3072 (da 3072 a 4071) e la mappa degli attributi dei caratteri (colore, luminosita', flash) va da 2048 a 3047. Nella mappa video sono contenuti i D/CODE (vedi Paragrafo 4.8) dei caratteri, che sono diversi per i caratteri in campo diretto e in campo inverso.
Abbiamo preparato il sottoprogramma HARD1, che legge con la funzione PEEK i codici D/CODE dei caratteri presenti sul video, li trasforma in codici ASCII, se i D/CODE sono maggiori di 128 (campo inverso) predispone il carattere di controllo per attivare sulla stampante il campo inverso, e li stampa, riproducendo i caratteri del video con i caratteri della stampante. HARD1 puo' quindi servire per copiare su carta un video in modo testo, rispettando il campo diretto e il campo inverso.

Nel programma COPIAVIDEO1 abbiamo usato HARD1 per ricopiare il video due volte, prima con il set maiuscolo/grafico e poi con il

set minuscolo/maiuscolo. Prima di far girare il programma, lo abbiamo listato con LIST sul video. Dato che il programma non e' molto lungo, esso entra tutto in un quadro video. Come puoi vedere dal risultato ogni linea e' di 40 caratteri; avresti ottenuto un risultato diverso trasferendo la lista alla stampante con CMD.

```
1 REM COPIAVIDEO1
2 REM COPIA SET MAIUSCOLO/GRAFICO
3 H1$=CHR$(145):GOSUB10000
4 H1$=CHR$(17):GOSUB10000:STOP
10000 REM HARD1
10001 OPEN4,4:PRINT#4
10002 H1=256*PEEK(1342)-40
10003 FORH0=0TO24:H0$=H1$:H1=H1+40
10004 FORH2=H1TOH1+39:H3=PEEK(H2)
10005 IFH3>128THENH3=H3-128:H4=1:H0$=H0$
+CHR$(18)
10006 IF(H3>0)*(H3<32)THENH3=H3+64:GOTO1
0010
10007 IF(H3>31)*(H3<64)THEN10010
10008 IF(H3>63)*(H3<96)THENH3=H3+128:GOT
O10010
10009 IF(H3>95)*(H3<128)THENH3=H3+64:GOT
O10010
10010 H0$=H0$+CHR$(H3)
10011 IFH4=1THENH0$=H0$+CHR$(146):H4=0
10012 NEXTH2:PRINT#4,H0$:NEXTH0
10013 PRINT#4:CLOSE4:RETURN

RUN
```

COMMENTO A COPIAVIDEO1

.3: pone H1$=CHR$(145) per attivare il set maiuscolo/grafico e chiama il sottoprogramma HARD1.

.4: pone H1$=CHR$(17) per attivare il set minuscolo/maiuscolo e chiama il sottoprogramma HARD1; poi si ferma.

COMMENTO A HARD1

.10001: apre la stampante.

.10002: prepara in H1 l'indirizzo di inizio della mappa video - 40.

.10003: inizia un ciclo per H0 da 0 a 24 per leggere le 25 linee del video. Pone H0$=H1$ e incrementa H1 di 40, per puntare a inizio riga; alla fine della lettura di ogni riga torna ad eseguire queste due ultime operazioni.

.10004: inizia il ciclo per leggere i 40 caratteri di una riga. Legge il D/CODE di un carattere con la funzione PEEK.

.10005: se il D/CODE supera 128, aggiunge alla stringa H0$ il codice 18 per ottenere RVS-ON sulla stampante e pone H4=1.

.10006/10009: trasforma il D/CODE in codice ASCII.

.10010: aggiunge alla stringa H0$ il nuovo carattere.

.10011: se H4=1 aggiunge alla stringa H0$ il codice 146 per tornare a RVS-OFF e pone H4=0.

.10012: torna alla lettura di un nuovo carattere fino alla conclusione del ciclo di carattere nella riga. Poi stampa la stringa H0$ sulla stampante e passa alla prossima riga video fino alla conclusione del ciclo di riga.

.10013: chiude la stampante e ritorna al programma principale.

Il programma COPIAVIDEO2 stampa sul video 3 stringhe in campo diretto e 3 in campo inverso, nel set maiuscolo/grafico, poi chiama HARD1 per ricopiarle sulla carta. Dopo cambia il set di caratteri, stampa le stesse stringhe e chiama HARD1 per eseguire la copia del video.

```
1 REM COPIAVIDEO2
2 PRINT"[graphic characters]";:PRINT"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
3 PRINT"[graphic characters]"
4 PRINT"[graphic characters]"
5 PRINT"[graphic characters]ABCDEFGHIJKLMNOPQRSTUVWXYZ[graphic characters]"
6 PRINT"[graphic characters]"
7 PRINT"[graphic characters]"
10 H1$=CHR$(145):GOSUB10000
11 H1$=CHR$(17):GOSUB10000:STOP
10000 REM HARD1
10001 OPEN4,4:PRINT#4
10002 H1=256*PEEK(1342)-40
10003 FORH0=0TO24:H0$=H1$:H1=H1+40
10004 FORH2=H1TOH1+39:H3=PEEK(H2)
10005 IFH3>128THENH3=H3-128:H4=1:H0$=H0$+CHR$(18)
10006 IF(H3>0)*(H3<32)THENH3=H3+64:GOTO10010
10007 IF(H3>31)*(H3<64)THEN10010
10008 IF(H3>63)*(H3<96)THENH3=H3+128:GOTO10010
10009 IF(H3>95)*(H3<128)THENH3=H3+64:GOTO10010
10010 H0$=H0$+CHR$(H3)
10011 IFH4=1THENH0$=H0$+CHR$(146):H4=0
10012 NEXTH2:PRINT#4,H0$:NEXTH0
10013 PRINT#4:CLOSE4:RETURN
```

RISULTATI COPIAVIDEO2

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
[graphic characters]
[graphic characters]
[reverse-video characters]
[reverse-video characters]
[reverse-video characters]

abcdefghijklmnopqrstuvwxyz
[graphic characters]
QWERTYUIOP[graphic characters]+IASDFGHJKL-ZXCVBNM
[reverse-video characters]
[reverse-video characters]
[reverse-video characters]
```

Il programma COPIAVIDEO3 usa il sottoprogramma HARD2 per copiare il video. In HARD2 la copia del video viene fatta aprendo il video come file, leggendo con GET# i caratteri, riga per riga, e stampandoli. Dato che i caratteri sono letti in codice ASCII, il programma non rispetta il campo inverso, salvo che per i caratteri contenuti tra virgolette.

```
1 REM COPIAVIDEO3
2 SA=0:GOSUB10000
3 SA=7:GOSUB10000
4 STOP
10000 REM HARD2
10005 PRINT"■";
10007 OPEN3,3:OPEN4,4,SA
10010 FORK=0TO24:A$=""
10015 FORJ=0TO39
10020 GET#3,B$:A$=A$+B$
10025 NEXTJ
10030 PRINT#4,A$
10035 NEXTK:PRINT
10040 CLOSE3:CLOSE4
10045 RETURN

RUN
```

```
1 rem copiavideo3
2 sa=0:gosub10000
3 sa=7:gosub10000
4 stop
10000 rem hard2
10005 print"■";
10007 open3,3:open4,4,sa
10010 fork=0to24:a$=""
10015 forj=0to39
10020 get#3,b$:a$=a$+b$
10025 nextj
10030 print#4,a$
10035 nextk:print
10040 close3:close4
10045 return

run
```

Prima di far girare il programma lo abbiamo listato sul video. Esso viene stampato nei due set di caratteri.

Affrontiamo ora il problema di ottenere su carta una copia identica del video. Il problema e' abbastanza complesso infatti noi disponiamo dei codici dei caratteri, in D/CODE se li leggiamo con la funzione PEEK. Dato che nel COMMODORE PLUS-4 la mappa dei caratteri in ROM descrive solo i caratteri in campo diretto, D/CODE da 0 a 127, per i due set, dobbiamo analizzare il codice, e, se maggiore di 128, andare a prendere la descrizione del codice corrispondente al campo diretto e scambiare i bit 0 con bit 1 e viceversa. Le due mappe di descrizioni dei due set di caratteri occupano ciascuna 1024 byte (128*8=1024) e si trovano in ROM da 53248 (D000H) a 54271, e da 54272 (D400H) a 55296. Per poter ricostruire i caratteri per punti sulla stampante occorrono le descrizioni dei caratteri, che pero' non sono accessibili da BASIC in ROM. In conseguenza i programmi BASIC che vogliono accedere alla descrizione dei caratteri, senza usare routine in linguaggio macchina, devono essere lanciati dopo:

.aver abbassato il top della memoria del BASIC,

.aver trasferito con l'istruzione MONITOR i 2K della ROM caratteri in RAM. L'istruzione MONITOR puo' accedere alla ROM senza problemi.

Abbiamo preparato il sottoprogramma HARD3, che lavora cosi':

.Preleva i D/CODE dal video con la funzione PEEK e li memorizza in una matrice X(M,I), dove M rappresenta il numero delle righe

video da ricopiare e I il numero dei caratteri per riga.

.Usa una matrice D a due indici, che abbia un numero di righe multiplo di 7, infatti per i caratteri grafici della stampante si devono usare 7 punti incolonnati, e che sia sufficiente a contenere le descrizioni dei caratteri di ogni riga video, ognuno formato da 8 linee. In tale matrice, che ha 40 colonne, vengono memorizzate le descrizioni di ogni carattere, occupando 8 righe. Le ultime righe della matrice rimangono eventualmente vuote, dato che si adatta una struttura multipla di 8 in una multipla di 7.

.Preleva dalla matrice D i byte a gruppi di 7 (sulla stessa colonna), calcola i codici degli 8 caratteri grafici a cui essi danno luogo (un carattere per ogni colonna di bit) e li memorizza in Z(7), poi li trasforma in stringa e li stampa. La prima riga stampata corrisponde quasi a una riga del video, infatti ha lavorato su 7 linee di punti invece che su 8, ma alla fine l'immagine e' formata dallo stesso numero di punti del video. Noterai una certa differenza con i caratteri stampati di solito dalla stampante.

Noi abbiamo richiamato il sottoprogramma HARD3 dal programma COPIAVIDEO4. Abbiamo limitato il numero di righe video da ricopiare per ridurre il tempo di esecuzione del programma. Ci siamo limitati a scrivere sul video i caratteri di D/CODE compreso tra 32 e 127, che sono 96 e quindi occupano circa 3 righe video. Abbiamo dimensionato in conseguenza le matrici X e D; X(2,39) per avere tre righe e 40 colonne, e D(27,39) per avere 28 righe e 40 colonne, infatti per 3 righe di caratteri ci vogliono 3x8=24 linee di punti e il multiplo di 7 piu' vicino a 24 e' 28.

```
1 REM COPIAVIDEO4
3 DIMX(2,39),D(27,39),Z(7)
50 H2$=CHR$(142)+CHR$(146)
55 GOSUB100:D=0:GOSUB10000
57 H2$=CHR$(142)+CHR$(18)
59 GOSUB100:D=0:GOSUB10000
70 H2$=CHR$(14)+CHR$(146)
75 GOSUB100:D=1:GOSUB10000
77 H2$=CHR$(14)+CHR$(18)
79 GOSUB100:D=1:GOSUB10000
81 PRINTCHR$(146)CHR$(142):STOP
100 PRINT"□";H2$;
105 FORK=32TO127:PRINTCHR$(K);:NEXTK
115 PRINT:RETURN
10000 REM HARD3
10005 OPEN4,4
10010 H1=256*PEEK(1342)-40
10015 FORH0=0TO2:H1=H1+40
10020 J=0:FORH2=H1TOH1+39
10021 X(H0,J)=PEEK(H2)
10060 J=J+1:NEXTH2:NEXTH0
10063 GOSUB20000
10064 GOSUB20020
```

```
10065 PRINT#4:CLOSE4:RETURN
20000 REM PRELEVA DESCRIZIONI
20005 A=14080+D*1024
20007 FORM=0TO2:FORI=0TO39:Y=X(M,I)
20008 SW=0:IFY>127THENSW=1:Y=Y-128
20009 FORK=0TO7:D(M*8+K,I)=PEEK(A+Y*8+K)
20011 IFSW=1THEND(M*8+K,I)=2↑8-1-D(M*8+K,I)
20015 NEXTK:NEXTI:NEXTM:RETURN
20020 PRINT#4,CHR$(8);
20035 FORN=0TO27STEP7
20038 FORI=0TO39
20039 FORL=0TO7:Z(L)=0:NEXTL
20040 FORM=0TO6:Y=D(M+N,I)
20043 FORL=0TO7
20045 IFINT(Y/2↑(7-L))=0THEN20060
20050 Z(L)=Z(L)+2↑M:Y=Y-2↑(7-L)
20060 NEXTL:NEXTM
20065 A$="":FORL=0TO7:A$=A$+CHR$(Z(L)+128)
20066 NEXTL
20067 PRINT#4,A$;:NEXTI:PRINT#4
20080 NEXTN:PRINT#4,CHR$(15):RETURN
```

RISULTATI COPIAVIDEO4

```
 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFG
HIJKLMNOPQRSTUVWXYZ[£]↑←
```

```
 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFG
HIJKLMNOPQRSTUVWXYZ[£]↑←
```

```
 !"#$%&'()*+,-./0123456789:;<=>?@abcdefg
hijklmnopqrstuvwxyz[£]↑←ABCDEFGHIJKLMNO
PQRSTUVWXYZ
```

```
 !"#$%&'()*+,-./0123456789:;<=>?@abcdefg
hijklmnopqrstuvwxyz[£]↑←ABCDEFGHIJKLMNO
PQRSTUVWXYZ
```

COMMENTO A COPIAVIDEO4

Prima di caricare o scrivere il programma e' necessario eseguire in immediato le seguenti istruzioni:

.1): POKE 55,255:POKE 56,54:CLR per abbassare il top della memoria BASIC a 14079 (36FFH).

.2): MONITOR per attivare il MONITOR

T D000 D7FF 3700 per trasferire i 2K delle descrizioni dei caratteri dalla ROM (53248) in RAM (14080).

X per uscire dal MONITOR.

Segue il commento al programma.
.3: dimensiona le 3 matrici:
X(2,39) per contenere i D/CODE di 3 righe video,
D(27,39) per contenere le descrizioni per punti delle 3 righe di caratteri, ma con un numero di linee di punti multiplo di 7.
Z(7), per contenere i codici dei caratteri grafici ottenuti da 7 byte incolonnati in D.
.50/55: predispone il set maiuscolo/grafico in campo diretto sul video, chiama il sottoprogramma in 100 per scrivere i caratteri sul video, pone D=0 per puntare al primo set e chiama HARD3 per eseguire la copia.
.57/59: come sopra ma per il campo inverso.
.70/75: come sopra, ma per il secondo set e il campo diretto.
.77/79: come sopra, ma per il campo inverso.
.81: ripristina il set iniziale e si ferma.
Ottieni come risultati 96 caratteri del primo set, prima in campo diretto e poi in campo inverso, e poi 96 caratteri del secondo set, ancora nei due modi.

COMMENTO A HARD3
.10005: apre la stampante.
.10010: prepara l'indirizzo della mappa video.
.10015/10060: trasferisce il contenuto della mappa video nella matrice X, usando la funzione PEEK.
.10063: chiama il sottoprogramma in 20000 per riempire la matrice D con le descrizioni dei caratteri.
.10064 chiama il sottoprogramma in 20020 per stampare la copia del video.
.10065: chiude la stampante e ritorna al programma principale.
.20000/20015: trasferisce le descrizioni dei caratteri nella matrice D; se il D/CODE supera 127 scambia tra loro i bit 0 con bit 1 e viceversa, per ottenere la rappresentazione in campo inverso.
.20020: inizia la parte stampa passando in modo grafico.
.20035: predispone l'analisi della matrice D in strisce di 7 linee.
.20038: predispone l'analisi dei 40 caratteri di una riga.
.20039: azzera il vettore Z(7).
.20040/20060: decodifica a gruppi i 7 byte della striscia, ottenendo i codici di 8 caratteri grafici.
.20065/20066: calcola la stringa corrispondente
.20067: stampa 8 caratteri grafici contenuti in A$ e passa al prossimo gruppo di byte. Alla fine della striscia va a capo.
.20080: chiude il ciclo di analisi delle strisce di D, e alla fine ripassa in modo testo ed esce.

A nostro avviso l'interesse di questo programma sta solo nelle difficolta' di programmazione incontrate, cioe' nell'aver fatto

lavorare il BASIC a livello di bit. Infatti il programma risulta piuttosto lento.
Nel prossimo paragrafo presentiamo un programma in linguaggio macchina che lavora sulla pagina grafica; si potrebbe prepararne uno analogo che lavora sul video in modo testo, riuscendo ad ottenere la copia molto piu' velocemente.

5.7 COPIA DEL VIDEO GRAFICO SU CARTA

La pagina grafica viene attivata passando in modo grafico con l'istruzione GRAPHIC. Essa inizia al byte 8192 ($2000) ed e' formata da 8000 byte, 8 per ogni posizione carattere del video in modo testo. I byte della pagina grafica sono utilizzati come segue:

.i primi 8 descrivono il primo insieme di 8x8 punti situato nell'angolo in alto a sinistra, corrispondente alla posizione carattere di coordinate 0,0;

.i successivi 8 byte descrivono l'insieme di 8x8 punti corrispondente alla posizione carattere di coordinate 1,0 (accanto e a destra della precedente), e cosi' via;

.gli ultimi 8 byte, di indirizzo da 16184 a 16191, descrivono l'ultimo insieme di 8x8 punti corrispondente alla posizione carattere di coordinate 39,24.

Nella Figura 5.1 riportiamo la corrispondenza tra i byte della pagina grafica e le posizioni sul video.

Per risolvere il problema della copia su carta, abbiamo gia' disponibile la descrizione per punti, ma dobbiamo trasformarla in caratteri grafici, cioe' in colonne di 7 punti. Abbiamo gia' eseguito questo lavoro in BASIC nel paragrafo precedente con il sottoprogramma HARD3. In quel caso nella matrice D(27,39) abbiamo messo noi i byte gia' organizzati nella sequenza nella quale danno il disegno del quadro, come appare nella Figura 5.1. In questo caso, invece, la sequenza di byte per indirizzo crescente e' utilizzata in un modo piu' complicato e dobbiamo trovare un opportuno algoritmo che ci permetta di riferirci ad essi. Per questa ragione, dopo attento esame, abbiamo preparato un sottoprogramma in ASSEMBLER, che chiamiamo HARD4; esso viene richiamato dal programma BASIC VIDEOGRAF e si ottiene la copia su carta della pagina grafica.

Vediamo ora quali algoritmi abbiamo usato per prelevare dalla pagina grafica i punti nella sequenza necessaria per la stampa. Dobbiamo costruire caratteri grafici per la stampante, formati da colonne di 7 punti, cioe' utilizzare i byte che danno il quadro video a strisce alte 7 punti. La prima volta, per la prima linea di stampa, da 8192 a 8198, da 8200 a 8206,..., da 8504 a 8510, e cosi' via. Dal momento che 200 diviso 7 da' 28 con resto di 4,

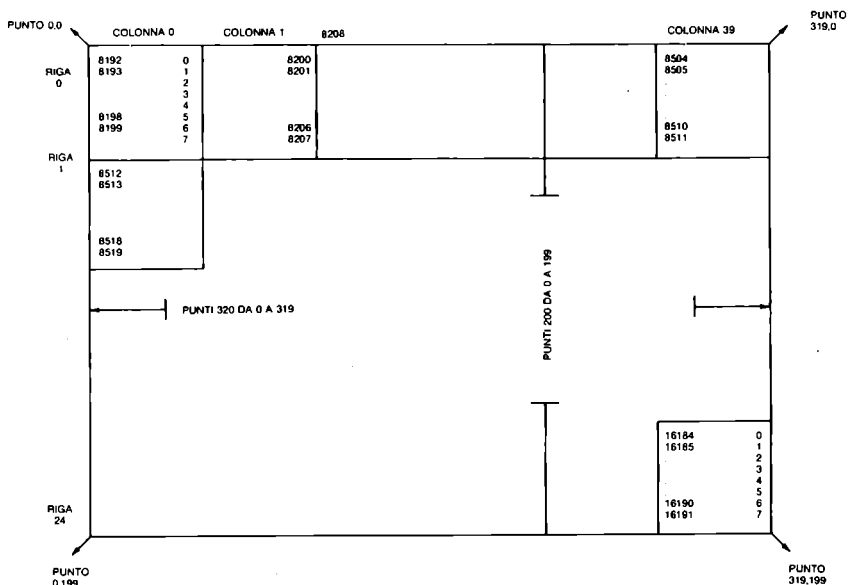



Figura 5.1 Come i byte della pagina grafica danno il quadro video

l'ultima striscia di caratteri grafici avra' solo 4 punti veri e gli ultimi 3, i piu' significativi a 0.
Consideriamo un singolo punto sul video; le sue coordinate grafiche X e Y danno la posizione, con 0<=X<=319 e 0<=Y<=199. Noi dobbiamo trovare un algoritmo che metta in relazione X e Y con l'indirizzo del byte (da 8192 a 16191) a cui il punto appartiene e con la posizione del bit corrispondente nel byte (da 0, a destra, a 7, a sinistra). Chiamiamo IB l'indirizzo del byte e IP la posizione del bit nel byte, espressa con il suo peso; abbiamo:

```
IB=8192+8*INT(X/8)+320*INT(Y/8)+(Y AND 7)

IP=2^(7 - (X AND 7))
```

Con riferimento alla Figura 5.1, verifichiamo le due formule per il bit di posizione 5 nel byte di indirizzo 8513, e per il bit di posizione 4 nel byte di indirizzo 16184.

.1) byte 8513, bit di posizione 5
coordinata X=2, infatti si trova nella terza colonna da sinistra

coordinata Y=9, infatti si trova nella decima linea di punti dall'alto

```
IB=8192+8*INT(2/8)+320*INT(9/8)+(9 AND 7)
  =8192+0+320*1+1
  =8192+320+1
  =8513
IP=2^(7-(2 AND 7))
  =2^(7-2)
  =2^5
```

.2) byte 16184, bit di posizione 4

coordinata X=315, infatti si trova nella quintultima posizione dell'ultimo byte della riga

coordinata Y=192, infatti si trova nella 193-esima linea di punti dall'alto

```
IB=8192+8*INT(315/8)+320*INT(192/8)+(192 AND 7)
  =8192+8*38+320*24+0
  =8192+312+7680

  =16184
IP=2^(7-(315 AND 7))
  =2^(7-3)
  =2^4
```

```
0 REM VIDEOGRAF
10 GRAPHIC 1:GRAPHIC 0
20 FORI=0TO244:READA:POKE5888+I,A:NEXT
30 OPEN4,4:PRINT#4,CHR$(8);
40 FORI=0TO321:PRINT#4,CHR$(192);:NEXT
50 CMD4:SYS5888
60 FORI=0TO321:PRINT#4,CHR$(129);:NEXT
61 PRINT#4,CHR$(14)
70 CLOSE4
999 REM DATI PER HARD4
1000 DATA169,7,141,71,63,169,0,141
1010 DATA72,63,169,255,32,210,255,169
1020 DATA0,141,64,63,141,65,63,169
1030 DATA128,141,74,63,169,0,141,73
1040 DATA63,173,72,63,24,109,73,63
1050 DATA141,68,63,32,121,23,238,73
1060 DATA63,173,73,63,205,71,63,208
1070 DATA232,173,74,63,32,210,255,238
1080 DATA64,63,173,64,63,208,3,238
1090 DATA65,63,201,64,208,201,173,65
1100 DATA63,201,1,208,194,169,255,32
1110 DATA210,255,169,13,32,210,255,173
1120 DATA72,63,24,105,7,141,72,63
1130 DATA201,196,208,5,169,4,141,71
1140 DATA63,173,68,63,201,199,208,146
1150 DATA96,173,64,63,41,248,141,66
1160 DATA63,173,64,63,41,7,141,67
1170 DATA63,173,68,63,74,74,74,141
1180 DATA69,63,173,68,63,41,7,141
1190 DATA70,63,169,32,133,4,169,0
1200 DATA133,3,172,69,63,240,16,165
1210 DATA3,24,105,64,133,3,165,4
1220 DATA105,1,133,4,136,208,240,165
1230 DATA3,24,109,66,63,133,3,165
```

```
1240 DATA4,109,65,63,133,4,173,70
1250 DATA63,24,101,3,133,3,165,4
1260 DATA105,0,133,4,169,128,174,67
1270 DATA63,240,4,74,202,208,252,33
1280 DATA3,240,17,169,1,172,73,63
1290 DATA240,4,10,136,208,252,13,74
1300 DATA63,141,74,63,96
```

COMMENTO A VIDEOGRAF

.10: sposta il programma in $4000 (vedi Paragrafo 2.8) per non sporcare con le variabili il sottoprogramma in linguaggio macchina, le memorie degli attributi per la pagina grafica e la pagina grafica stessa (che occupano rispettivamente i byte da $1700 a $17F4, da $1800 a 1BE7, da 1C00 a 1FE7 e da $2000 a $3F40).

.20: carica il programma in linguaggio macchina in memoria.

.30: apre il canale con la stampante e la pone in modo grafico.

.40: disegna la striscia di contorno superiore.

.50: dirotta l'output sulla stampante e esegue la routine in linguaggio macchina.

.60: disegna la striscia di contorno inferiore e fa uscire la stampante dal modo grafico.

.70: chiude il canale con la stampante.

.1000/1300: dati relativi a HARD4.

Vediamo ora il programma ASSEMBLER HARD4:

```
MONITOR

   PC   SR AC XR YR SP
; 0000 00 00 00 00 F8

. 1700  A9 07     LDA #$07
. 1702  8D 47 3F  STA $3F47
. 1705  A9 00     LDA #$00
. 1707  8D 48 3F  STA $3F48
. 170A  A9 FF     LDA #$FF
. 170C  20 D2 FF  JSR $FFD2
. 170F  A9 00     LDA #$00
. 1711  8D 40 3F  STA $3F40
. 1714  8D 41 3F  STA $3F41
. 1717  A9 80     LDA #$80
. 1719  8D 4A 3F  STA $3F4A
. 171C  A9 00     LDA #$00
. 171E  8D 49 3F  STA $3F49
. 1721  AD 48 3F  LDA $3F48
. 1724  18        CLC
. 1725  6D 49 3F  ADC $3F49
. 1728  8D 44 3F  STA $3F44
. 172B  20 79 17  JSR $1779
. 172E  EE 49 3F  INC $3F49
. 1731  AD 49 3F  LDA $3F49
. 1734  CD 47 3F  CMP $3F47
. 1737  D0 E8     BNE $1721
. 1739  AD 4A 3F  LDA $3F4A
. 173C  20 D2 FF  JSR $FFD2
. 173F  EE 40 3F  INC $3F40
. 1742  AD 40 3F  LDA $3F40
. 1745  D0 03     BNE $174A
. 1747  EE 41 3F  INC $3F41
. 174A  C9 40     CMP #$40
. 174C  D0 C9     BNE $1717
. 174E  AD 41 3F  LDA $3F41
. 1751  C9 01     CMP #$01
. 1753  D0 C2     BNE $1717
. 1755  A9 FF     LDA #$FF
. 1757  20 D2 FF  JSR $FFD2
. 175A  A9 0D     LDA #$0D
. 175C  20 D2 FF  JSR $FFD2
. 175F  AD 48 3F  LDA $3F48
. 1762  18        CLC
. 1763  69 07     ADC #$07
. 1765  8D 48 3F  STA $3F48
. 1768  C9 C4     CMP #$C4
. 176A  D0 05     BNE $1771
. 176C  A9 04     LDA #$04
. 176E  8D 47 3F  STA $3F47
. 1771  AD 44 3F  LDA $3F44
. 1774  C9 C7     CMP #$C7
. 1776  D0 92     BNE $170A
. 1778  60        RTS
```

```
. 1779   AD 40 3F  LDA $3F40     . 17B9   18         CLC
. 177C   29 F8     AND #$F8      . 17BA   6D 42 3F   ADC $3F42
. 177E   8D 42 3F  STA $3F42     . 17BD   85 03      STA $03
. 1781   AD 40 3F  LDA $3F40     . 17BF   A5 04      LDA $04
. 1784   29 07     AND #$07      . 17C1   6D 41 3F   ADC $3F41
. 1786   8D 43 3F  STA $3F43     . 17C4   85 04      STA $04
. 1789   AD 44 3F  LDA $3F44     . 17C6   AD 46 3F   LDA $3F46
. 178C   4A        LSR           . 17C9   18         CLC
. 178D   4A        LSR           . 17CA   65 03      ADC $03
. 178E   4A        LSR           . 17CC   85 03      STA $03
. 178F   8D 45 3F  STA $3F45     . 17CE   A5 04      LDA $04
. 1792   AD 44 3F  LDA $3F44     . 17D0   69 00      ADC #$00
. 1795   29 07     AND #$07      . 17D2   85 04      STA $04
. 1797   8D 46 3F  STA $3F46     . 17D4   A9 80      LDA #$80
. 179A   A9 20     LDA #$20      . 17D6   AE 43 3F   LDX $3F43
. 179C   85 04     STA $04       . 17D9   F0 04      BEQ $17DF
. 179E   A9 00     LDA #$00      . 17DB   4A         LSR
. 17A0   85 03     STA $03       . 17DC   CA         DEX
. 17A2   AC 45 3F  LDY $3F45     . 17DD   D0 FC      BNE $17DB
. 17A5   F0 10     BEQ $17B7     . 17DF   21 03      AND ($03,X)
. 17A7   A5 03     LDA $03       . 17E1   F0 11      BEQ $17F4
. 17A9   18        CLC           . 17E3   A9 01      LDA #$01
. 17AA   69 40     ADC #$40      . 17E5   AC 49 3F   LDY $3F49
. 17AC   85 03     STA $03       . 17E8   F0 04      BEQ $17EE
. 17AE   A5 04     LDA $04       . 17EA   0A         ASL
. 17B0   69 01     ADC #$01      . 17EB   88         DEY
. 17B2   85 04     STA $04       . 17EC   D0 FC      BNE $17EA
. 17B4   88        DEY           . 17EE   0D 4A 3F   ORA $3F4A
. 17B5   D0 F0     BNE $17A7     . 17F1   8D 4A 3F   STA $3F4A
. 17B7   A5 03     LDA $03       . 17F4   60         RTS
```

Questa routine usa alcuni byte come memoria di lavoro:

.$3F40,$3F41: contatore per X (byte basso/alto).
.$3F42: INT(X/8)*8 (byte basso).
.$3F43: X AND 7.
.$3F44: Y.
.$3F45: INT(Y/8).
.$3F46: Y AND 7.
.$3F47: numero di linee che compongono la striscia corrente (normalmente 7; per l'ultima striscia 4).
.$3F48: 7 per numero di strisce gia' stampate.
.$3F49: numero di linea nella striscia.
.$3F4A: codice del carattere da stampare.

Dal momento che questo sottoprogramma non e'molto semplice, riportiamo nelle Figure 5.2 e 5.3, rispettivamente il diagramma a blocchi di HARD4 e quello della routine interna ad esso.

Per facilitare la comprensione del programma, abbiamo numerato ogni blocco dei diagrammi (con un numero posto fuori a destra) e diamo la corrispondenza tra i blocchi e le linee del listato ASSEMBLER:

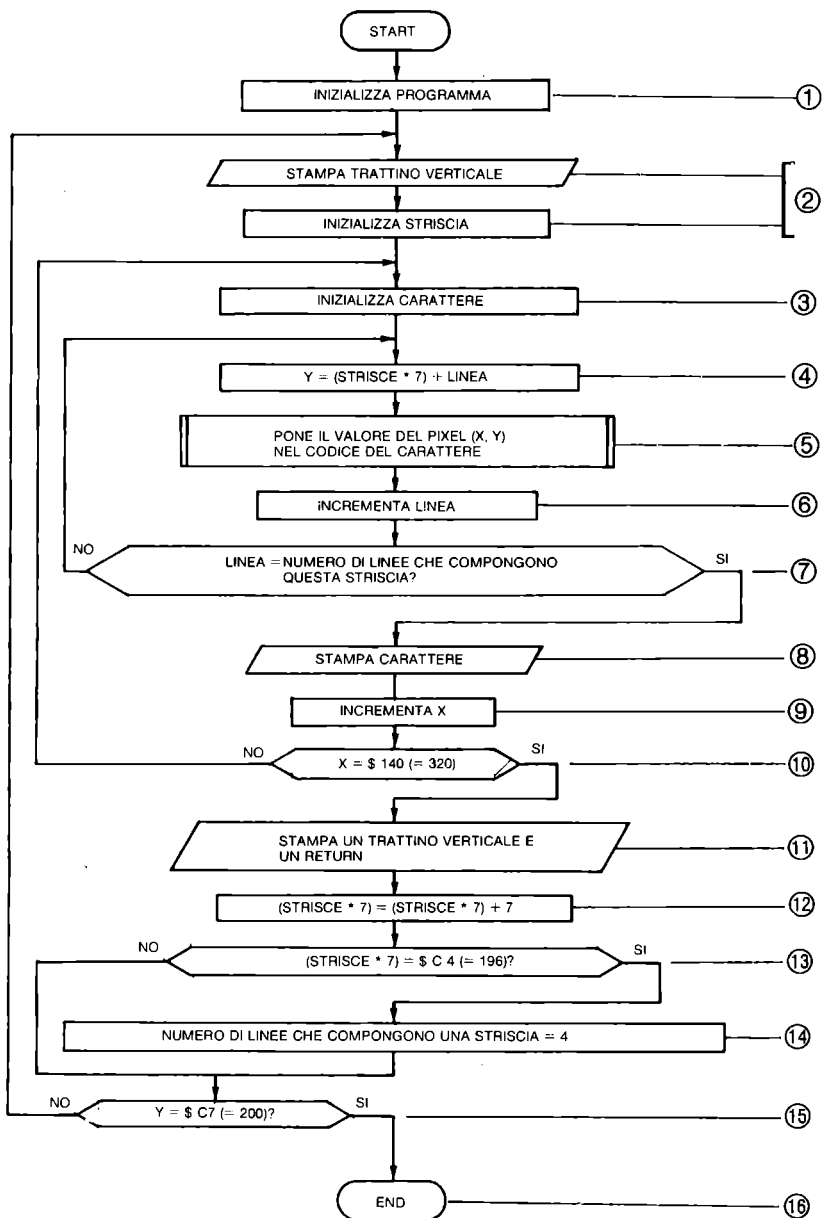



Figura 5.2 Diagramma a blocchi sottoprogramma HARD4

Per la Figura 5.2:
BLOCCO 1 : linee 1700÷1707.
BLOCCO 2 : linee 170A÷1714.
BLOCCO 3 : linee 1717÷171E.

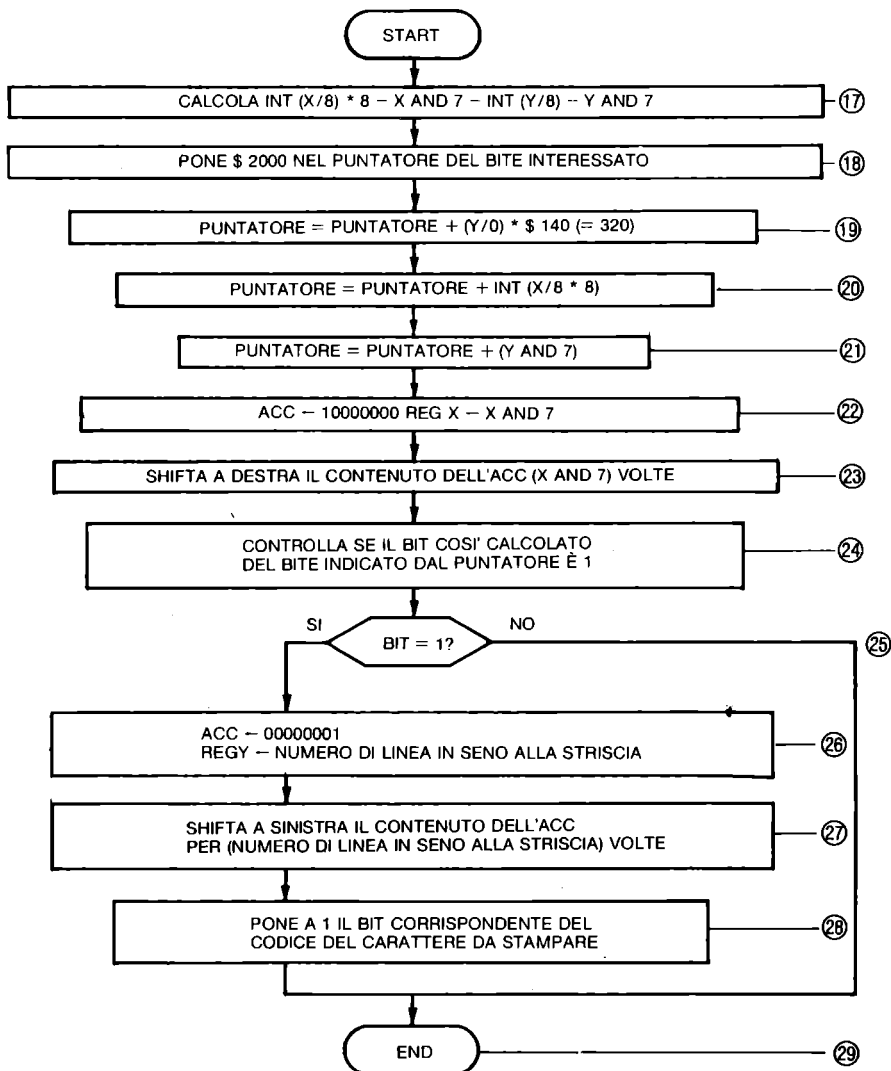



Figura 5.3 Diagramma a blocchi sottoprogramma interno a HARD4

BLOCCO 4 : linee 1721-1728.
BLOCCO 5 : linea 172B.
BLOCCO 6 : linea 172E.
BLOCCO 7 : linee 1731-1737.
BLOCCO 8 : linee 1739-173C.
BLOCCO 9 : linee 173F-1747.
BLOCCO 10 : linee 174A-1753.
BLOCCO 11 : linee 1755-175C.
BLOCCO 12 : linee 175F-1765.
BLOCCO 13 : linee 1768-176A.
BLOCCO 14 : linee 176C-176E.
BLOCCO 15 : linee 1771-1776.
BLOCCO 16 : linea 1778.

Per la Figura 5.3:
BLOCCO 17 : linee 1779-1797.
BLOCCO 18 : linee 179A-17A0.
BLOCCO 19 : linee 17A2-17B5.
BLOCCO 20 : linee 17B7-17C4.
BLOCCO 21 : linee 17C6-17D2.
BLOCCO 22 : linee 17D4-17D6.
BLOCCO 23 : linee 17D9-17DD.
BLOCCO 24 : linee 17DF.
BLOCCO 25 : linee 17E1.
BLOCCO 26 : linee 17E3-17E5.
BLOCCO 27 : linee 17E8-17EC.
BLOCCO 28 : linee 17EE-17F1.
BLOCCO 29 : linee 17F4.

Per poter usare con soddisfazione VIDEOGRAF, devi prima aver disegnato qualcosa sulla pagina grafica, ed essere tornato in modo testo. Noi abbiamo usato il programma ES2.16. Il tasto ESC seguito da un numero modifica il numero dei caratteri di una linea video.
Con ES2.16 abbiamo preparato il disegno che vedi riprodotto da HARD4. Ovviamente puoi usare qualunque programma per produrre un disegno in pagina grafica.

RISULTATO VIDEOGRAF

CAPITOLO 6

I FILE SU DISCO

6.1 INTRODUZIONE

Al COMMODORE PLUS-4 possono essere collegate in serie diverse periferiche tra unita' a disco e stampanti.
Le unita' disco sono di norma vendute con dn=8, cioe' gli switch interni sono posizionati su 8. Se il calcolatore e' collegato a piu' unita', la prima puo' mantenere dn=8, la o le altre possono avere dn da 9 a 11. Per modificare il "dn", si puo' agire all'interno dell'unita', rendendo permanente il nuovo numero, oppure procedere all'assegnazione temporanea via software del nuovo numero, al momento dell'accensione.
Noi ci occupiamo dell'unita' 1541 e i programmi esempio si riferiscono al collegamento di una sola unita'.

L'unita' 1541 e' una periferica intelligente, cioe' essa lavora in modo indipendente dopo aver ricevuto i comandi dal calcolatore. Le sue parti componenti sono:
. un microprocessore 6502,
. 16K di memoria ROM, contenenti il DOS (Disk Operating System),
. 2K di memoria RAM per i buffer e le memorie di lavoro,
. un'interfaccia seriale IEEE488,
. due porte di comunicazione,
. le parti elettromeccaniche necessarie.
Le due porte di comunicazione consentono di collegare in serie piu' unita'.
Nella parte anteriore dell'unita' sono visibili:
. lo sportellino e la fessura per l'introduzione del dischetto,
. un indicatore luminoso, a sinistra, a luce verde, che indica se l'unita' e' accesa,
. un indicatore luminoso, piu' verso il centro, a luce rossa, che indica, quando acceso, se e' in corso fisicamente un'operazione sul dischetto. Questo indicatore lampeggia quando si verifica un errore.

Il dischetto flessibile, floppy, e' contenuto in una busta protettiva di plastica con alcune aperture, visibili nella Figura 6.1.

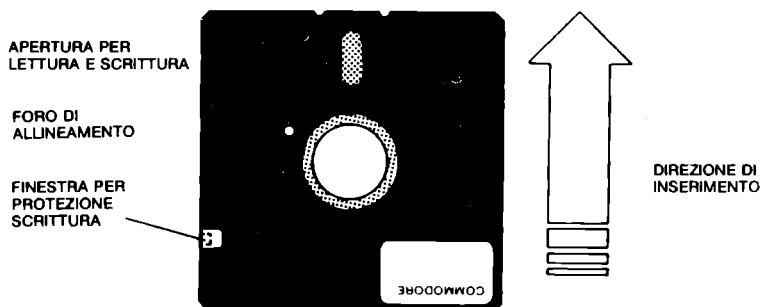



Figura 6.1 Il dischetto nella sua busta

Il dischetto non deve mai essere maneggiato toccando le aperture. La testina di lettura e scrittura agisce attraverso la finestra oblunga centrale. La finestra laterale serve per proteggere il disco da scrittura, basta chiuderla con una delle apposite etichette. Il floppy non deve essere estratto con acceso l'indicatore rosso.

Le registrazioni sul floppy sono eseguite secondo tracce concentriche. Sul tipo di floppy usati per l'unita' 1541 sono disponibili 40 tracce su una sola faccia; di queste ne vengono utilizzate solo 35. Ogni traccia e' divisa in blocchi, chiamati settori, in numero diverso per gruppi di tracce, come sotto riportato.

FORMATO DEL FLOPPY

Numero traccia	Numero settori	Numerazione settori
da 1 a 17	21	da 0 a 20
da 18 a 24	19	da 0 a 18
da 25 a 30	18	da 0 a 17
da 31 a 35	17	da 0 a 16

Facendo i conti risultano disponibili in tutto:
17*21 + 7*19 + 6*18 + 5*17 = 683 settori.
Il settore e' il record fisico del floppy; esso contiene 256 byte. Nel seguito indichiamo con byte 0 il primo e con byte 255

l'ultimo del settore. La traccia 18, di 19 settori, che si trova al centro del dischetto (vedi Figura 6.2), e' utilizzata in un modo particolare; essa contiene l'indice delle registrazioni effettuate sul floppy e la mappa dell'occupazione dei settori. Questo e' necessario dato che il floppy e' un supporto di registrazione ad accesso diretto, non rigorosamente sequenziale come il nastro, e si deve conoscere l'ubicazione delle diverse registrazioni.

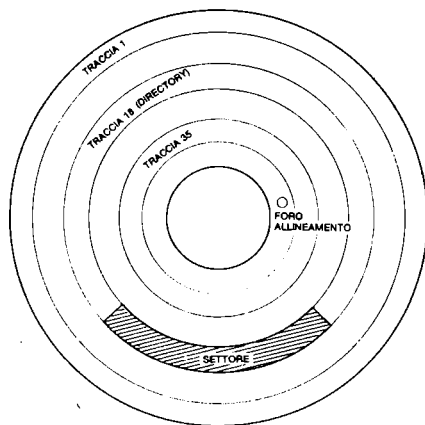



Figura 6.2 Schema non in scala del dischetto

Il dischetto nuovo non puo' essere utilizzato immediatamente; prima e' necessario eseguire su di esso un'operazione di preparazione all'uso, che si chiama "formattazione". Essa consiste nella registrazione sul dischetto degli indirizzi di traccia e settore, nell'assegnazione di un nome e di una identificazione di 2 caratteri, nella creazione della "mappa" del dischetto e della "directory".

Vediamo l'utilizzo della traccia 18 e dei suoi 19 settori, di indirizzo da 0 a 18.

TRACCIA 18 - SETTORE 0

. byte da 0 a 143, mappa di occupazione, chiamata BAM (Block Availability Map).
Questi 144 byte sono utilizzati cosi':
. byte 0 e 1, concatenamento al settore seguente, contengono 18

(12H) e 1 (01H), traccia 18 settore 1.

. byte 2, contiene 65, codice ASCII della lettera A, che indica il formato dell'unita'.

. byte 3, contiene tutti bit 0 e non e' usato.

. byte da 4 a 143, 140 byte che contengono la mappa di occupazione, utilizzando 4 byte per ogni traccia (35*4=140). Ogni quartina di byte e' utilizzata cosi':

. primo byte, numero dei settori ancora disponibili nella traccia;

. secondo byte, mappa dei primi 8 settori della traccia, utilizzando i bit a partire dal meno significativo; quello piu' a destra per il settore 0, quello piu' a sinistra per il settore 7;

. terzo byte, mappa degli 8 settori seguenti, da 8 a 15;

. quarto byte, mappa degli ultimi settori della traccia, a partire dal 16. Restano inutilizzati alcuni bit a seconda delle tracce.

Se il bit e' 1, il settore corrispondente e' libero, se e' 0, esso e' occupato.

. byte da 144 a 255, identificazione del dischetto, blocco di inizio della directory:

. byte da 144 a 161, 18 byte, di cui 16 per il nome, se esso e' piu' corto viene completato con il codice ASCII 160, corrispondente al carattere SHIFT-spazio, gli ultimi 2 byte sempre con codice 160;

. byte 162 e 163, i due caratteri di identificazione (ID) del floppy;

. byte 164, codice 160;

. byte 165 e 166, caratteri 2A, versione del DOS e formato dischetto;

. byte da 167 a 170, codice 160;

. byte da 171 a 255, non utilizzati, con tutti i bit a 0.

TRACCIA 18 - SETTORI DA 1 A 18

Contengono l'indice del disco "directory"; in ogni settore e' registrato l'indice di 8 file. In conseguenza il numero massimo di file registrabili e' 144, 18*8=144. Ogni registrazione nell'indice viene chiamata "entrata" (entry), e occupa 30 byte. La struttura di ogni settore e' la seguente:

. byte 0 e 1, concatenamento al settore seguente. Se il settore e' l'ultimo della catena, il byte 0 contiene tutti bit 0 e il byte 1 contiene il puntatore all'ultimo byte utilizzato, quindi 255 se il settore e' completo.

. byte da 2 a 31, entrata 1:

. primo byte, tipo del file:

tutti bit 0 per DEL, file cancellato,

128 + 1 per SEQ, file sequenziale,

128 + 2 per PRG, file programma,

128 + 3 per USR, file utente,
128 + 4 per REL, file relativo.

. secondo e terzo byte, indirizzo di traccia e settore del primo blocco del file.

. dal quarto al 19-esimo byte, 16 byte per il nome del file, completato con codice 160 se piu' corto.

. 20-esimo e 21-esimo byte, usati per i file relativi, contengono gli indirizzi di traccia e settore del primo blocco "side sector" (indice interno del file relativo).

. 22-esimo byte, usato solo per i file relativi, contiene la lunghezza del record, <=254.

. dal 23-esimo al 26-esimo byte, 4 byte non usati.

. 27-esimo e 28-esimo byte, traccia e settore del primo blocco del file rimemorizzato, cioe' del file che e' stato memorizzato usando il carattere "@".

. 29-esimo e 30-esimo byte, numero dei blocchi occupati dal file, nell'ordine LO-HI.

. byte 32 e 33, 2 caratteri separatori, contenenti tutti bit 0.
. byte da 34 a 63, entrata 2.
. ...
. ...
. byte 224 e 225, 2 caratteri separatori.
. byte da 226 a 255, entrata 8.

Dei 683 settori disponibili, 19 sono usati per le informazioni sopra descritte; in conseguenza restano disponibili per i file 664 settori (683-19=664).

Dopo l'inserimento del dischetto nell'unita' 1541, al primo accesso avviene l'operazione di "inizializzazione", che consiste nel:

. allineare la testina di lettura con la traccia,
. leggere il settore 0 della traccia 18 in uno dei buffer di 256 byte dell'unita'. Questa operazione e' essenziale, infatti per poter scrivere sul dischetto deve essere consultabile la BAM, altrimenti si rischia di scrivere sopra settori gia' occupati. Inoltre la BAM deve essere riscritta sul dischetto, prima di toglierlo dall'unita', altrimenti il dischetto non e' piu' utilizzabile in modo corretto.

Devi fare attenzione e non confondere la formattazione con l'inizializzazione; le differenze tra le due operazioni sono:

. la formattazione agisce sul dischetto e lo modifica,
. l'inizializzazione preleva informazioni dal dischetto e le carica nella RAM dell'unita' 1541.

La directory, invece, viene letta a pezzi quando serve, settore per settore, per trovare la localizzazione di un file e le informazioni necessarie alla sua gestione. Essa viene parzialmente riscritta per registrare le modifiche, quando serve.

Abbiamo detto che in un settore si possono registrare 256 byte; in realta' il settore e' formato anche da altri byte, ma questi sono utilizzati dal sistema, per l'indirizzamento e per i controlli. Tu puoi usare solo i 256 byte disponibili per l'utente.

Le caratteristiche tecniche del floppy sono:
.Capacita' totale: 174848 byte (683*256)
.Capacita' per file sequenziali 168656 byte (in ogni settore i primi 2 byte sono usati per il concatenamento, quindi 254*664)
.Capacita' per file relativi 167132 (664*256 - 664*2(concatenamento) - 254*6(side sector)), con al massimo 65535 record logici per file
.Entrate nella directory: 144
.Settori per traccia: da 17 a 21
.Byte per blocco: 256
.Tracce: 35
.Totale blocchi: 683, di cui disponibili 664 per registrare file.

Le operazioni di lettura e scrittura relative al floppy avvengono sempre a livello di blocco fisico, cioe' di un intero settore.

6.2 IL DOS

Il DOS (Disk Operating System) e' una versione del sistema operativo DOS residente nella ROM dell'unita' 1541; esso esegue i comandi che riceve dal calcolatore, funzionando in modo indipendente. Il COMMODORE PLUS-4 invia attraverso l'interfaccia seriale le sequenze di byte alla periferica 1541; il DOS riconosce se si tratta di comandi o di dati e gestisce le operazioni disco.

Nel Paragrafo 4.1 abbiamo riepilogato le istruzioni BASIC per la gestione dei file; ora vediamo il significato dei parametri per la periferica 1541.
. lfn, numero logico del file puo' variare da 0 a 127.
. dn, numero dell'apparecchiatura, vale 8 se si usa una sola unita' collegata. Puo' variare da 8 a 11 per piu' unita' collegate.
. sa, secondo la terminologia COMMODORE viene in questo caso chiamato "canale" invece di "indirizzo secondario". Esso gioca un ruolo molto importante, infatti il suo valore permette al DOS di interpretare i dati trasmessi. I valori possibili per "sa" vanno da 0 a 15:
  . sa=15, canale comandi;
  . sa=0, canale usato per aprire un file in lettura;
  . sa=1, canale usato per aprire un file programma in scrittura;

. sa=2,...,14, canale per trasferimento dati.
. nomef, nome del file puo' essere lungo fino a 16 caratteri. Esso puo' essere preceduto dal numero del drive, che e' 0 e puo' essere omesso per unita' singole, deve essere 0 o 1 per unita' doppie.
. tipo, tipo del file, puo' essere: PRG, SEQ, USR, REL (abbreviato alla prima lettera).
. modo, puo' essere W per scrivere e R per leggere, nel caso dei file sequenziali. Si puo' usare anche A (Append) con il significato di prolungare (aggiungere in coda dati) a un file sequenziale.

Attraverso il canale 15, il calcolatore invia all'unita' 1541, con le istruzioni OPEN e/o PRINT#, stringhe di comandi, che esaminiamo nel seguito. Il DOS provvede a interpretare i comandi e li manda in esecuzione; esso assegna al canale lo spazio di memoria RAM necessario per lavorare e i buffer necessari. La RAM dell'unita' 1541 comprende 8 buffer di 256 byte ciascuno; di essi 4 sono sempre impegnati per la BAM, le variabili di lavoro e il controllo delle operazioni. Restano a disposizione solo 4 buffer per i file; in conseguenza si possono gestire contemporaneamente 3 o 4 file, a seconda del loro tipo.

Prendiamo ora in esame un primo gruppo di stringhe-comando per la gestione del dischetto; esse si possono inviare al DOS in BASIC, sia in modo immediato che da programma. Per inviare il comando si possono seguire due strade:

.1) OPENlfn,8,15
    PRINT#lfn,stringa-comando

.2) OPENlfn,8,15,stringa-comando

e in ambedue i casi devi terminare con CLOSElfn. Se non si esegue la CLOSE, ad una successiva apertura si ha errore. Puoi decidere di aprire all'inizio del programma il canale 15, di inviare i comandi con PRINT#lfn e di chiudere il canale solo alla fine del programma.

I comandi di utilita' generale disponibili, che si possono scrivere per esteso o abbreviandoli alla prima lettera, sono:

.NEW, per formattare il dischetto,
.INITIALIZE, per allineare la testina e caricare la BAM,
.VALIDATE, per sistemare la BAM in base alle entrate valide registrate nella directory,
.COPY, per copiare file,
.RENAME, per cambiare nome a un file,

.SCRATCH, per cancellare un file.

Il BASIC 3.5 del COMMODORE PLUS-4 mette a disposizione alcuni comandi che producono lo stesso effetto di alcuni di questi, e non hanno bisogno di essere preceduti dalla OPEN del canale 15.

NEW
puo' essere usato per:
.preparare un disco nuovo (o gia' usato) per l'uso. Vengono registrati gli indirizzi di traccia e settore, il nome e l'identificazione, viene preparata la BAM.
.cancellare un disco gia' usato, mantenendo invariata l'dentificazione, ma senza riscrivere gli indirizzi, e aggiornando la BAM.
La stringa si scrive: "Ndr:nome,XX"
.dr, puo' essere omesso se 0.
.nome, e' il nome da assegnare al dischetto, massimo 16 caratteri.
.XX; e' la "identificazione" del dischetto; deve essere di 2 caratteri (scelti a piacere). Omettendo XX con un floppy gia' usato, si ottiene di cancellare il precedente contenuto, senza riscrivere gli indirizzi. Non puoi omettere XX con un floppy nuovo.

Il comando BASIC equivalente e' HEADER.

INITIALIZE
deve essere usato per allineare la testina di lettura e scrittura all'inizio della traccia e per caricare la BAM in un buffer dell'unita' 1541.
La stringa si scrive: "Idr"
.dr, puo' essere omesso se 0.
Quando inserisci il floppy nell'unita' questa operazione deve avvenire automaticamente, comunque e' bene eseguirla quando si usa molto il dischetto. Prima di eseguirla devi chiudere i file eventualmente aperti.
Non esiste un comando BASIC equivalente.

VALIDATE
serve per rimettere in ordine un dischetto nel quale la situazione della BAM non corrisponde alle entrate della directory, ci sono file non correttamente chiusi (con un asterisco di fianco al tipo nella lista della directory). Devi eseguire questa operazione quando la somma dei blocchi occupati e dei blocchi liberi non da' 664.
La stringa si scrive: "Vdr"
.dr, puo' essere omesso se 0.
Il comando rigenera la BAM in base alle entrate della directory e

cancella le entrate della directory non valide.
Il comando BASIC equivalente e' COLLECT.

COPY
consente di ottenere copie di un file di qualunque tipo. Se il floppy e' unico (come per l'unita' 1541) il nome vecchio e il nome nuovo devono essere diversi. Si possono anche fondere piu' file sequenziali di dati in un unico file. Lavora su unita' singola o su unita' a due drive, non su due unita' diverse.
La stringa si scrive:
"Cdr:dnomef=dr:snomef" per copia di un file
"Cdr:dnomef=dr:snomef1,snomef2,..." per fondere file sequenziali di dati.
.dnomef, e' il file destinazione.
.snomef, e' il file sorgente.
Il file sorgente deve essere stato correttamente chiuso.
Il comando BASIC equivalente e' COPY.

Inoltre per unita' a due floppy in BASIC e' disponibile anche il comando BACKUP (DUPLICATE del DOS).
Per il momento non e' disponibile un'unita' a 2 drive da collegare direttamente al COMMODORE PLUS-4.

RENAME
serve per cambiare nome a un file nella directory senza spostare il file. La stringa si scrive: "Rdr:nnomef=vnomef"
.nnomef, e' il nome nuovo.
.vnomef, e' il nome vecchio.
Il file su cui operi deve essere stato correttamente chiuso.
Il comando BASIC equivalente e' RENAME.

SCRATCH
cancella uno o piu' file ponendo DEL (tutti bit 0) come tipo nell'entrata della directory e aggiorna la BAM.
La stringa si scrive:
"Sdr:nomef" per un solo file
"Sdr:nomef1,nomef2,.." per piu' file.Ti consigliamo di cancellare i file che non ti servono piu' prima di riscriverli e di non usare il carattere "@" nella OPEN o nelle SAVE, DSAVE. Infatti quest'ultimo modo di procedere puo', a volte, generare degli inconvenienti sul dischetto.
Il comando BASIC equivalente e' SCRATCH.

Per tutti i comandi visti il "dn" viene citato nella OPEN e seleziona l'unita', se ci sono piu' collegamenti.

Oltre a quelli gia' esaminati sono disponibili altri tre gruppi di comandi che servono per:

.gestire direttamente le operazioni di lettura e scrittura sul dischetto accedendo a un blocco fisico, individuato dal suo indirizzo di traccia e settore.
.gestire i file relativi.
.aggiungere al DOS routine utente, che possono essere lette dal floppy e scritte sul floppy e mandarle in esecuzione.

GESTIONE DIRETTA

Esaminiamo ora i comandi per la gestione diretta delle operazioni sul floppy. Per poter eseguire questi comandi deve essere aperto il canale 15 e deve essere aperto uno dei canali, da 2 a 14, per il trasferimento dei dati.
Nel seguito presupponiamo che siano state eseguite le due OPEN, prima della PRINT# che invia il comando, e che alla fine delle operazioni vengano eseguite le due corrispondenti CLOSE.
Le OPEN devono essere eseguite nel seguente ordine:
. OPENlfn,8,15 per aprire il canale comandi con un determinato "lfn". Noi, per abitudine usiamo il numero 15 per questo "lfn", cosi' non dobbiamo ricordare un altro numero: OPEN15,8,15.
. OPENlfn,8,sa,"#" per aprire il canale per i dati. Lo "lfn" usato qui deve essere diverso da quello della OPEN precedente; esso deve variare tra 0 e 127; "sa" puo' variare da 2 a 14. Noi di solito usiamo per "lfn" e "sa" valori uguali, compresi tra 2 e 14, ma escludiamo il 4, dato che, per abitudine, lo usiamo per la stampante. Il carattere "#" deve essere presente e significa apertura per accesso diretto. I buffer dell'unita' 1541 sono numerati da 0 a 7; se vuoi puoi scrivere dopo il carattere "#" un numero per scegliere un buffer particolare, con il rischio di trovarlo gia' occupato. Se scrivi "#4", chiedi di usare il quinto buffer per i dati. Qualora desideri sapere quale dei buffer il sistema ha usato, puoi subito dopo la OPEN..."#", eseguire GET#lfn,A$; in A$ trovi il numero del buffer assegnato. Questa ultima OPEN non produce una registrazione nella directory; questo fatto puo' essere pericoloso, infatti, se si eseguono i comandi VALIDATE o COLLECT, le registrazioni dirette spariscono, nel senso che vengono liberati i settori della BAM non registrati nella directory.

Al termine delle operazioni la CLOSE del canale dei dati deve essere eseguita prima della CLOSE del canale comandi. Il canale comandi puo' essere aperto all'inizio del programma e chiuso alla fine.

I comandi disponibili in questo gruppo sono:

. BLOCK-ALLOCATE, per allocare un blocco, abbreviato in B-A;
. BLOCK-FREE, per liberare un blocco, abbreviato in B-F;
. BLOCK-READ, per leggere un blocco nel buffer dei dati, abbreviato in B-R;

. BLOCK-WRITE, per scrivere il contenuto del buffer dei dati sul disco;
. BUFFER-POINTER, per posizionare il puntatore in una determinata posizione nel buffer dei dati;
. USER1, simile a B-R, abbreviato in U1;
. USER2, simile a B-W, abbreviato in U2.
Nelle spiegazioni che seguono usiamo i parametri gia' noti, e, in piu', "t" per traccia e "s" per settore, "p" per posizione del puntatore nel buffer.
Precisiamo come si svolgono le operazioni dirette sul floppy:
.SCRITTURA: le istruzioni PRINT#lfn,lista-dati scrivono nel buffer dei dati nella RAM dell'unita' 1541, facendo avanzare il puntatore; per trasferire il blocco sul dischetto si usa l'apposito comando trasmesso sul canale 15, con PRINT#.
.LETTURA: il blocco viene trasferito nel buffer dei dati nella RAM dell'unita' 1541 dall'apposito comando, trasmesso sul canale 15, con PRINT#; le istruzioni INPUT#lfn,lista-dati e GET#lfn,lista-dati leggono i dati dal buffer, facendo avanzare il puntatore.
La posizione del puntatore nel buffer puo' essere controllata, usando il relativo comando, trasmesso sul canale 15.

BLOCK-ALLOCATE, si scrive:
   "B-A:"dr;t;s
registra nella BAM l'occupazione del blocco di indirizzo t,s. E' importante allocare un settore libero, altrimenti si cancella qualcosa. Per essere sicuri che un settore e' libero si chiede di allocarlo e subito dopo si analizza la situazione di errore tramite il canale 15. L'errore 65: NO BLOCK, segnala che il settore e' gia' occupato. In questo caso le due variabili ET e ES, fornite dall'analisi dell'errore, danno l'indirizzo t,s del primo settore disponibile, che puo' essere allocato. Se, invece, ET contiene 0, questo significa che il dischetto e' pieno. EN contiene 0 se l'allocazione del blocco non ha creato problemi. E' importante tener presente che il sistema non evita di allocare settori eventualmente liberi nella traccia 18 della directory; per questa ragione devi controllare con il tuo programma di non andare ad occuparli. Infatti un'eventuale estensione della directory per nuove entrate potrebbe poi danneggiare le tue registrazioni (vedi routine ALLOCA nel Paragrafo 6.8).

Vedi NOTA al comando seguente.

BLOCK-FREE, si scrive:
   "B-F:"dr;t;s
libera il blocco di indirizzo t,s e aggiorna la BAM.
NOTA: B-A e B-F lavorano anche se prima non e' stata eseguita una OPEN..."#", ma questo e' pericoloso, infatti la BAM viene riscritta sul floppy quando si chiude un canale per la trasmis-

sione dei dati; se esso non e' stato aperto si puo' danneggiare il dischetto.

BLOCK-READ, si scrive:
"B-R:"sa;dr;t;s
trasferisce il settore di indirizzo t,s nel buffer dei dati assegnato al momento della OPEN..."#" con lo stesso valore di "sa". Nella posizione 0 del buffer (primo carattere) viene trovato il valore del puntatore (fine record) al momento della registrazione del blocco. Tale indicazione e' importante perche' in fase di lettura dal buffer con le istruzioni INPUT# o GET# la parola di stato ST contiene 64 quando si raggiunge la posizione di fine record.

BLOCK-WRITE, si scrive:
"B-W:"sa;dr;t;s
scrive il contenuto del buffer dei dati, assegnato al momento della OPEN..."#" con lo stesso valore di "sa", nel blocco individuato dall'indirizzo t,s. Registra nella posizione 0 il valore raggiunto dal puntatore e predispone il puntatore sulla posizione 1. Se il blocco viene letto con B-R, il primo carattere disponibile e' quello di posizione 1, mentre, se la lettura viene effettuata con U1, il primo carattere disponibile e' quello di posizione 0.

BUFFER-POINTER, si scrive:
"B-P:"sa;p
consente di spostare il puntatore all'interno del buffer dei dati. La posizione del puntatore e' molto importante, infatti le istruzioni di I/O agiscono a partire dalla sua posizione. Nelle due coppie di istruzioni che scrivono o leggono un settore abbiamo indicato dove si trova il puntatore dopo la loro esecuzione. Questo comando consente di controllare completamente la posizione del puntatore.
Il sistema gestisce i file sequenziali usando i primi due caratteri di ogni blocco, di posizione 0 e 1, per concatenare tra loro i settori; quando usi i file random puoi mantenere questa organizzazione preoccupandoti tu di riempire i primi due caratteri e posizionando i dati a partire dalla posizione 2. In sostanza in un settore possono cosi' essere utilizzati solo 254 byte. Lo spostamento del puntatore consente di raggiungere qualunque campo nel record di un file random.

USER1, si scrive:
"U1:"sa;dr;t;s
agisce come B-R, con la differenza che legge tutto il settore, senza tener conto della posizione del puntatore al momento della scrittura. Dopo l'esecuzione di U1 il puntatore si trova nella posizione 0 dell buffer.

USER2, si scrive:
  "U2:"sa;dr;t;s

agisce come B-W, ma non registra in posizione 0 la posizione finale del puntatore.

Nella descrizione delle stringhe comando noi abbiamo usato il formato: comando abbreviato, seguito da due punti, tra virgolette, e dopo i parametri separati da ";". E' possibile scrivere anche tutto tra virgolette, nel caso che i parametri siano costanti, usando il separatore virgola; per esempio, cosi':
  "U2:3,0,15,8"
I comandi possono essere scritti anche in modo non abbreviato.

GESTIONE FILE RELATIVI

I file relativi sono formati da record logici di lunghezza fissa. Il DOS li gestisce ricevendo una stringa comando che indica il numero d'ordine del record logico nel file e la posizione del puntatore nel record logico.
Anche in questo caso sono necessarie 2 istruzioni OPEN:
 . OPENlfn,8,15, per aprire il canale comandi.
 . OPENlfn,8,sa,nomef+",L,"+CHR$(LU), per aprire il canale dei dati con "lfn" diverso da quello del canale comandi, "sa" compreso tra 2 e 14, L parametro richiesto per segnalare che l'apertura riguarda un file relativo, seguito dalla lunghezza del record logico passata come stringa. La lunghezza del record logico e' al massimo 254. Questa OPEN produce la registrazione di un'entrata per il file nella directory.
Al termine delle operazioni la CLOSE del canale dei dati deve essere eseguita prima di quella relativa al canale dei comandi.
Il comando disponibile, da trasmettere con una PRINT# sul canale 15, prima di ogni operazione di lettura o scrittura relativa al buffer, serve per selezionare il record logico e per posizionare il puntatore al suo interno.

Esso si scrive:
  "P"+CHR$(sa)+CHR$(LO)+CHR$(HI)+CHR$(BI)
 .P, significa puntatore.
 .CHR$(sa), fornisce il numero del canale usato nella OPEN del canale dei dati, come stringa.
 .CHR$(LO)+CHR$(HI), forniscono il numero del record logico che si vuole trattare nella forma byte basso-byte alto. Si calcolano cosi':
  HI=INT(numerorecord/256)
  LO=numerorecord-HI*256.
 .CHR$(BI), fornisce il puntatore nel record. BI=1 per il primo campo, BI=N per puntare al campo che inizia dopo i primi N-1 caratteri.

Dopo aver selezionato un record e una posizione, devi scrivere con una sola PRINT#,lista-dati tutti i campi che desideri nel record. Se esegui altre PRINT#,lista-dati, senza riposizionarti, vai a scrivere sui record successivi. Un record logico puo' essere scritto a pezzi, ma ogni volta devi riposizionarti al record e al campo desiderato.
Per leggere devi posizionarti al record e al campo desiderato e poi eseguire la INPUT#, lista-dati relativa a tutti i campi del record che ti interessano. Se non ripeti l'operazione di posizionamento con successive INPUT# leggi il file in modo sequenziale, un record dopo l'altro.
L'istruzione GET#, legge carattere per carattere a partire dalla posizione selezionata.

PROGRAMMAZIONE IN LINGUAGGIO MACCHINA

I comandi disponibili in questo gruppo sono:

. BLOCK-EXECUTE, per eseguire un programma in linguaggio macchina, memorizzato in un settore del floppy;
. Memory-Write, per registrare al massimo 34 byte nella RAM dell'unita' 1541;
. Memory-Read, per leggere il contenuto di byte della memoria dell'unita' 1541;
. Memory-Execute, per eseguire codice macchina a partire da un byte della memoria dell'unita' 1541;
. USERi, per saltare a particolari locazioni della memoria dell'unita' 1541 ed eseguire routine in codice macchina.

Questi comandi, salvo BLOCK-EXECUTE, richiedono solo la preventiva apertura del canale 15. Il comando BLOCK-EXECUTE richiede anche l'apertura di un altro canale con OPEN#lfn,8,sa,"#", per rendere disponibile un buffer per i dati. Nelle spiegazioni che seguono, oltre ai parametri gia' noti, usiamo anche:
.i, come numero di riferimento nella tabella dei salti USER.
.adl, byte basso dell'indirizzo del blocco di memoria.
.adh, byte alto dell'indirizzo del blocco di memoria.
.nc, numero caratteri da trasferire, da 1 a 34.
.dati, istruzioni in codice macchina; devi usare la funzione CHR$ con argomento uguale al numero decimale che rappresenta il contenuto di ogni byte.

BLOCK-EXECUTE, si scrive:
"B-E:"sa,dr,t,s
consente di caricare in un buffer dell'unita' 1541 il contenuto del blocco di indirizzo t,s, e di mandarlo in esecuzione, come programma in linguaggio macchina, a partire dalla posizione 0. Perche' questo comando possa essere eseguito con successo devono essere verificate le seguenti condizioni:

. la routine in linguaggio macchina deve trovarsi nel settore specificato e la prima istruzione deve stare nel primo byte (posizione 0);
. la routine deve occupare al massimo 256 byte;
. il settore del floppy deve essere stato scritto con il comando U2, per non danneggiare la posizione 0;
. la routine deve terminare logicamente con l'istruzione RTS.
La routine in linguaggio macchina puo' lavorare modificando o leggendo i dati degli altri buffer della RAM dell'unita' 1541.
Il comando puo' anche essere scritto senza abbreviazione, e i parametri, se costanti, possono stare tra le virgolette separati da virgola.

Memory-Write, si scrive:
   "M-W:"adl adh;nc;dati
consente di registrare, al massimo 34 byte, nella RAM dell'unita' 1541. I parametri adl e adh, che forniscono l'indirizzo di memorizzazione, si devono calcolare; per scrivere all'indirizzo 1794:
HI=INT(1794/256)=7
LO=1794-7*256=1794-1792=2
CHR$(2)+CHR$(7) forniscono la stringa da usare nel comando per "adl adh".
Il parametro "nc", deve essere passato come numero, o come variabile numerica.
I parametri "dati" sono ottenuti usando la funzione CHR$ avente come argomento i valori decimali dei byte da scrivere in memoria. Se con questo comando memorizzi una routine in linguaggio macchina, per eseguirla devi usare il comando Memory-Execute. Nota la differenza con BLOCK-EXECUTE.
Questo comando deve essere scritto in modo abbreviato.

Memory-Read, si scrive:
   "M-R:"adl adh
consente di leggere il contenuto del byte di indirizzo "adl adh" dalla memoria dell'unita' 1541.
L'indirizzo deve essere passato con la funzione CHR$ dei byte LO e HI.
Il contenuto del byte indirizzato viene reso disponibile sul canale 15, per leggerlo devi eseguire GET# sul canale 15. Se esegui piu' GET# successive l'indirizzo del byte viene incrementato automaticamente.
Per esempio:

```
OPEN15,8,15
PRINT#15,"M-R:"CHR$(X)CHR$(Y)
FORk=0TO7:GET#15,a$(k):NEXTk
CLOSE15
```

legge nella matrice a$(k) 8 byte a partire dall'indirizzo di memoria 256*Y+X dell'unita' 1541.

Il comando puo'essere scritto solo abbreviato.

Memory-Execute, si scrive:
"M-E:"adl adh
consente di eseguire una routine in codice macchina memorizzata a partire dal byte di indirizzo "adl adh". Per il parametro indirizzo vale quanto detto per M-R. Il comando puo' essere scritto solo in modo abbreviato.
Invece di prelevare ed eseguire una routine in codice macchina memorizzata su un settore del floppy, usando il comando BLOCK-EXECUTE, puoi usare prima il comando M-W per memorizzare la routine in RAM e poi M-E per eseguirla.

USERi, si scrive:
"Ui" con i=0,3,4,...,8,9 oppure:
"Ua" con a=J,C,D,...,H,I
consente di saltare a indirizzi fissi nella memoria dell'unita' 1541. Tali indirizzi sono:

U3	o	UC	salto a 1280	(0500H)
U4	o	UD	" " 1283	(0503H)
U5	o	UE	" " 1286	(0506H)
U6	o	UF	" " 1289	(0509H)
U7	o	UG	" " 1292	(050CH)
U8	o	UH	" " 1295	(050FH)
U9	o	UI	" " 65530	(FFFAH)
U0	o	UJ	" alla routine di accensione.	

Puoi memorizzare a partire da questi indirizzi, se sono RAM, routine in linguaggio macchina e andarle a eseguire usando i comandi di Ui relativi.

6.3 GESTIONE DEGLI ERRORI DISCO

Quando si verifica un errore relativo ad un'operazione dell'unita' 1541 l'indicatore luminoso a luce rossa comincia a pulsare, ma non si ha una segnalazione automatica di errore. Per rilevare gli errori devi inserire nei tuoi programmi una routine di errore, leggendo, attraverso il canale 15, i 4 dati seguenti:
. EN, numero dell'errore, dato numerico;
. EM$, messaggio di errore, dato di tipo stringa;
. ET, numero della traccia interessata, dato numerico;
. ES, numero del settore interessato, dato numerico.
Le 4 variabili citate non sono riservate, puoi usare quelle che vuoi, le nostre sono abbastanza mnemoniche. Puoi anche usare variabili di tipo stringa per tutti i dati.
Supponendo che all'inizio del programma sia stata eseguita l'istruzione: OPEN15,8,15, la routine di errore puo' avere questa forma:

```
5000 INPUT#15,EN,EM$,ET,ES
5002 IFEN=0 THEN RETURN
5004 PRINT"ERRORE: ";EN,EM$,ET,ES
5006 STOP
```

essa lascia aperto il canale 15. Ti raccomandiamo di eseguire il controllo degli errori dopo ogni operazione disco.
Un altro modo per controllare se ha avuto luogo un errore e' quello di leggere le due variabili riservate del BASIC: DS e DS$, dopo l'esecuzione di ogni operazione disco. In DS trovi il numero del messaggio di errore, in DS$ trovi il numero e il messaggio di errore, il numero della traccia e il numero del settore. Con DS$ ottieni le stesse informazioni fornite dalla routine di errore, solo che esse si trovano in una sola stringa. Per analizzare o stampare il contenuto di queste due variabili riservate non e' necessario il canale 15; esse sono automaticamente disponibili per il BASIC.
L'esecuzione della routine di errore o l'analisi delle variabili DS e DS$ hanno l'effetto di spegnere l'indicatore di errore se esso e' acceso.
Non devi confondere le segnalazioni di errore, relative ad operazioni disco, fornite dal sistema operativo del COMMODORE PLUS-4, con quelle del DOS. Se, per esempio, esegui un'istruzione PRINT# su un file non aperto, la segnalazione di errore dipende dal fatto che il COMMODORE PLUS-4 non ha trovato aperto il file nella tabella di gestione, il DOS non c'entra.

Il codice di errore 00 significa che l'operazione e' andata bene; il codice 01 che e' stato cancellato un file.

Inoltre, dopo ogni operazione disco puo' essere analizzata la variabile riservata ST; essa puo' avere i seguenti valori:

- 0 per tutto bene
- 1 per scrittura con tempizzazione errata
- 2 per lettura con tempizzazione errata
- 64 per segnalazione di EOF (End Of File)
- 128 per apparecchiatura non presente

e quindi non fornisce informazioni complete sull'andamento dell'operazione.

MESSAGGI ERRORE DOS

20 READ ERROR
il controllore del disco non riesce a trovare la testata del settore richiesto, o l'indirizzo non e' valido o il floppy e' rovinato.

21 READ ERROR
il controllore del disco non riesce a trovare il carattere di sincronizzazione sulla traccia richiesta. Puo' essersi verifi-

cato un errore di allineamento, un errore hardware o il floppy non essere formattato.

22 READ ERROR
per un comando DOS di tipo BLOCK non viene trovato il settore; puo' essere invalido l'indirizzo o danneggiato il floppy.

23 READ ERROR
si verifica un errore di CHECKSUM dopo la lettura di un blocco, cioe' i controlli sui caratteri letti non tornano; puo' dipendere da una messa a terra difettosa.

24 READ ERROR
si verifica un errore hardware, puo' dipendere da messa a terra difettosa.

25 WRITE ERROR
i dati registrati non corrispondono al contenuto del buffer.

26 WRITE PROTECT ON
tentativo di scrivere su un floppy con protezione, cioe' con finestrella laterale chiusa.

27 READ ERROR
errore nella lettura di una testata, puo' dipendere da messa a terra difettosa.

28 WRITE ERROR
non viene trovato, dopo la scrittura di un blocco, il carattere di sincronizzazione del blocco seguente. Il floppy puo' essere rovinato o non formattato o verificarsi un errore hardware.

29 DISK ID MISMATCH
non riesce a identificare un floppy, che puo' essere non formattato o rovinato.

30 SYNTAX ERROR
il DOS non riesce a interpretare un comando ricevuto, possono essere errati o mancare alcuni parametri.

31 SYNTAX ERROR
il DOS non riconosce un comando, per esempio esso inizia con uno spazio.

32 SYNTAX ERROR
comando troppo lungo, piu' di 58 caratteri.

33 SYNTAX ERROR
e' stato usato un nome non valido.

34 SYNTAX ERROR
il DOS non riconosce il nome di un file.

39 SYNTAX ERROR
non viene riconosciuto un comando inviato sul canale 15.

50 RECORD NOT PRESENT
tentativo di leggere un file dopo aver raggiunto EOF. Per i file relativi si verifica questo errore nella fase di preestensione e deve essere ignorato.

51 OVERFLOW IN RECORD
si tenta di scrivere in un blocco piu' caratteri di quelli consentiti.

52 FILE TOO LARGE
si scrivono piu' record di quelli previsti su un file relativo.

60 WRITE FILE OPEN
si apre in lettura un file gia' aperto per scrivere e non chiuso.

61 FILE NOT OPEN
si tenta un'operazione su un file non aperto.

62 FILE NOT FOUND
il file richiesto non viene trovato.

63 FILE EXISTS
si cerca di creare un file che esiste gia' e non si e' usato il carattere "@".

64 FILE TYPE MISMATCH
il tipo di operazione fa riferimento a un file di tipo diverso da quello esistente.

65 NO BLOCK
indica che il blocco richiesto con B-A e' gia' occupato, ma fornisce in ET e ES l'indirizzo del primo blocco libero. ET e ES sono a zero se il floppy e' pieno.

66 ILLEGAL TRACK AND SECTOR
si tenta di accedere a settori che non esistono.

67 ILLEGAL SYSTEM T OR S
traccia o settore non consentiti.

70 NO CHANNEL
il canale richiesto e' occupato o non ci sono canali liberi.

71 DIRECTORY ERROR
i controlli non consentono di creare una BAM valida. Probabilmente il dischetto non e' ricuperabile.

72 DISK FULL
indica o che il dischetto e' pieno o che sono esaurite le 144 entrate nella directory.

73 DOS MISMATCH
le versioni del DOS non sono compatibili in scrittura. Si tenta di scrivere su un floppy formattato con un'altra versione.

74 DRIVE NOT READY
non c'e' il floppy oppure esso non e' stato formattato o il drive e' guasto.

6.4 FILE DI PROGRAMMA

I programmi sono memorizzati sul dischetto come file di tipo PRG. Nella relativa entrata della directory si trovano tutte le informazioni necessarie per reperire il file, e cioe' l'indirizzo di traccia e settore del primo blocco, il numero dei blocchi occupati, e, se necessario, l'indirizzo del primo blocco per la rimemorizzazione con il carattere "@". Per analizzare il contenuto della directory puoi usare il programma DCOMEFS, riportato nel Paragrafo 6.8. I blocchi nei quali viene registrato un programma sono concatenati tra loro tramite i primi 2 caratteri, che recano il numero della traccia e del settore successivo. L'ultimo blocco reca 0 per numero di traccia e il valore del puntatore all'ultimo byte occupato al posto del numero del settore. Per vedere come il programma viene conservato sul disco puoi usare il programma TRAC/SET, riportato nel Paragrafo 6.8; solo che la lettura del contenuto dei blocchi non e' semplice, dato che si vedono i numeri in esadecimale. Se te ne occupi troverai che il programma termina con 2 byte contenenti tutti bit 0, dopo il byte a 0 che chiude l'ultima istruzione.

Per memorizzare un programma sul floppy sono disponibili le istruzioni SAVE e DSAVE, gia' esaminate nel precedente volume. Queste istruzioni possono essere scritte anche facendo precedere al nome del programma i caratteri "@:" oppure "@dr:", con l'effetto di cancellare la precedente versione del programma. Non ti consigliamo di procedere cosi'; infatti a volte operazioni di questo tipo danneggiano la BAM. Se devi rimemorizzare un programma e' preferibile prima cancellarlo con il comando SCRATCH, usato in una delle due versioni disponibili, e poi memorizzarlo con SAVE, ma senza usare il carattere di cancellazione "@".

Dopo la memorizzazione di un programma e' bene verificare con VERIFY la bonta' della registrazione. Puoi anche richiamare con il comando DIRECTORY la directory sul video; questo comando e' molto comodo dal momento che non cancella il contenuto della memoria e il programma presente resta invariato. Se invece carichi e listi la directory con:
OPEN15,8,15:LOAD"$",8:LIST il programma viene cancellato.

Per caricare in memoria un programma puoi usare i comandi DLOAD e LOAD. Se li usi in immediato il programma viene solo caricato, mentre se li usi da programma, ottieni anche la partenza, come se venisse eseguito: GOTO prima-linea. In questo caso ottieni il concatenamento dei programmi, ma il sistema non mette a posto il puntatore all'inizio della zona variabili, che varia in dipendenza dalla lunghezza dei programmi. Per operare correttamente il programma concatenato deve avere una prima istruzione che sistema tali puntatori, cosi':

```
0 POKE 45,PEEK(157):POKE 46,PEEK(158):CLR.
```

Con opportuni accorgimenti e' possibile eseguire programmi concatenati che abbiano le variabili in comune. Basta che il primo programma sposti il puntatore all'inizio delle variabili in posizione tale che vada bene per il programma piu' lungo e che tutti i programmi usino le stesse variabili con lo stesso significato.

Nel precedente volume abbiamo preso in esame il significato del flag che puo' essere aggiunto in fondo alle istruzioni per memorizzare e caricare i programmi e influisce sulla loro rilocabilita' in memoria.

6.5 FILE SEQUENZIALI DI DATI

I file sequenziali di dati su disco hanno le stesse caratteristiche di quelli su nastro; l'unica differenza rilevante consiste nel fatto che e' possibile aggiornare un file senza doverlo trascrivere tutto in memoria (vedi Paragrafo 11.6 del primo volume), ma creando un nuovo file sullo stesso dischetto, con nome diverso, nel quale trascrivere e aggiornare record dopo record il vecchio file. Al termine dell'aggiornamento il vecchio file puo' essere cancellato, e al nuovo file puo' essere assegnato il vecchio nome con il comando RENAME.
Riepiloghiamo le caratteristiche di questo tipo di file:
. record fisico di dimensioni pari a un settore, con i primi due caratteri usati per il concatenamento dei settori e gli altri 254 per i dati. La gestione del record fisico risulta trasparente per l'utente che lavora a livello di record logico. Alla fine del file viene riconosciuta la condizione EOF; essa puo' essere

controllata tramite la parola di stato ST, le variabili riservate DS o DS$, la routine di errore leggendo EN, EM$, ET e ES sul canale 15.

. record logico di lunghezza fissa o variabile, ma con lo stesso numero di campi, se si desidera poter leggere a livello di record. Tieni presente che se scrivi un file sequenziale con record logici tali che l'ultimo blocco fisico contiene 254 caratteri, cioe' e' completo al momento della chiusura, viene occupato un settore in piu' solo in modo apparente; cioe' tale settore manca nel computo dei blocchi liberi e occupati. Se esegui il comando COLLECT, o VALIDATE, la directory torna in ordine e il file puo' essere letto senza problemi.

. campi di lunghezza fissa o variabile, tenendo presenti le limitazioni imposte dal comando INPUT#, che non puo' leggere piu' di 88 caratteri tra due CHR$(13) (RETURN). Inoltre va tenuto presente che, se il separatore (registrato) tra i campi e' la virgola, essi devono essere letti tutti da una sola istruzione INPUT#, che lavora sui dati compresi tra due CHR$(13). Una variabile che contiene solo spazi da' luogo a un campo nullo, cioe' due caratteri separatori vicini, che in fase di lettura sono riconosciuti come uno solo, e quindi viene perso un campo.

. in fase di scrittura i caratteri separatori tra i dati della lista agiscono come per la cassetta: il punto e virgola non aggiunge spazi, la virgola si.

. al momento della OPEN per scrivere viene creata la nuova entrata nella directory; essa viene completata al momento della CLOSE. Se il file non viene chiuso, l'entrata rimane in stato irregolare e il file non puo' essere utilizzato.

. non e' necessario aprire il canale 15, se non vuoi gestire gli errori tramite di esso, ma ti raccomandiamo di farlo. Infatti e' buona norma eseguire la routine di errore dopo ogni operazione disco.

Le istruzioni BASIC disponibili sono:

```
OPENlfn,dn,sa,"dr:nomef,tipo,modo"
```

per aprire il file. I parametri possono essere costanti e variabili:

. lfn, numero logico del file da 0 a 127.
. dn, 8 per la prima unita'.
. sa, canale per i dati da 2 a 14.
. dr, numero drive, 0 per default, puo' essere omesso.
. nomef, nome del file, massimo 16 caratteri.
. tipo, S, che significa sequenziale.
. modo, W per scrivere e R per leggere. A per aprire in scrittura un file preesistente da allungare; dopo questa apertura il

primo comando di scrittura va a scrivere a partire dalla fine del vecchio file.
Puoi usare per la OPEN per scrivere questo formato:

OPENlfn,dn,sa,"@dr:nomef,S,W"

ottenendo di cancellare, se esiste, un file con lo stesso nome. Ti consigliamo di non usare questo formato, ma di cancellare il vecchio file con il comando SCRATCH e poi di aprirlo per scrivere.
Il DOS interpreta il comando e predispone il messaggio di errore in seguito all'esecuzione.
Possono essere aperti contemporaneamente fino a 3 file sequenziali su un floppy; il DOS li distingue in base al valore di "lfn".

PRINT#lfn,lista-dati

per scrivere i dati sul file aperto per scrivere, con numero logico "lfn". Non ripetiamo le considerazioni su lista-dati, gia' viste all'inizio.

INPUT#lfn,lista-dati

per leggere nelle variabili di lista-dati dal file aperto per leggere, con lo stesso valore di "lfn". Il tipo delle variabili deve concordare con il tipo dei dati che si leggono, altrimenti si ha errore.

GET#lfn,lista-dati

per leggere i dati carattere per carattere dal file aperto per leggere, con lo stesso valore di "lfn". E' meglio che lista-dati contenga solo variabili stringa.

CLOSElfn

per chiudere il file aperto con numero logico "lfn". La chiusura di un file aperto per scrivere fa aggiungere la segnalazione di EOF e provoca l'aggiornamento della directory e della BAM sul floppy. Per tutti i tipi di file la CLOSE provoca l'aggiornamento in memoria della tabella di gestione dei file.

Come esempio abbiamo realizzato il programma SEQDISCO. Esso gestisce un archivio sequenziale di dati su disco, consentendo le seguenti operazioni:

.1) creazione ex-novo del file. I record devono essere forniti in ordine in base ai primi due campi, l'ordine viene controllato e non accetta record con i primi due campi uguali.

.2) lista completa del file su stampante, nell'ordine di registrazione dei record.

.3) aggiornamento del file, cioe':

. inserimento nuovi record, che devono essere forniti in ordine.

. modifica record esistenti, procedendo nell'ordine di registrazione dei record.

. cancellazione record, procedendo nell'ordine di registrazione dei record.

L'archivio viene mantenuto in ordine crescente in base ai primi due campi di ogni record, ma senza ricorrere a ordinamento; quindi i record devono essere caricati in questo ordine e il programma scarta quelli fuori ordine.
Il programma e' stato scritto ricorrendo alla tecnica dei sottoprogrammi; in conseguenza esso e' facilmente modificabile per adattarlo alle proprie esigenze.
Noi abbiamo lavorato con un record logico di lunghezza variabile, ma formato da un numero fisso di campi, 5 per ogni record, ognuno di lunghezza variabile. I campi sono separati tra loro dal carattere CHR$(13); questo ci consente di avere per ogni campo la massima lunghezza possibile, cioe' 88 caratteri. La gestione dell'archivio avviene a livello record, trattando sempre lo stesso numero di campi; in conseguenza e' necessario che ogni record abbia tutti i campi stabiliti. Per mantenere questa caratteristica, quando un campo manca esso viene registrato con il carattere CHR$(160), che corrisponde a SHIFT-spazio, e che in fase di lettura non da' luogo a un campo vuoto, come succederebbe con il carattere spazio normale.
Nella Figura 6.3 riportiamo uno schema a blocchi della fase di aggiornamento del file, che risulta la piu' complicata. Durante questa fase viene creato un file temporaneo, di nome TEMP, sul quale viene aggiornato il file vecchio; alla fine il file vecchio viene cancellato e al file TEMP viene cambiato il nome, assegnandogli quello del vecchio file.

Il programma non presenta un menu' iniziale, ma pone successivamente domande sulle operazioni che si vogliono fare; a tali domande devi rispondere con S per SI e N per NO. Per uscire dalla richiesta di un record devi rispondere con il carattere asterisco "*" al primo campo. Se manca un campo puoi rispondere solo con RETURN, provvede il programma a sostiuirlo con il carattere CHR$(160) (SHIFT-spazio). Ogni fase termina con lo STOP; se vuoi proseguire devi eseguire il comando RUN. All'inizio di ogni fase viene chiesto il nome del file. Il file puo' essere lungo a piacere, compatibilmente allo spazio disponibile sul floppy. Nella fase di lista i dati sono presentati ponendo su una riga i primi due campi e sull'altra gli altri 3, inoltre non viene usato un conta righe per il cambio del foglio.

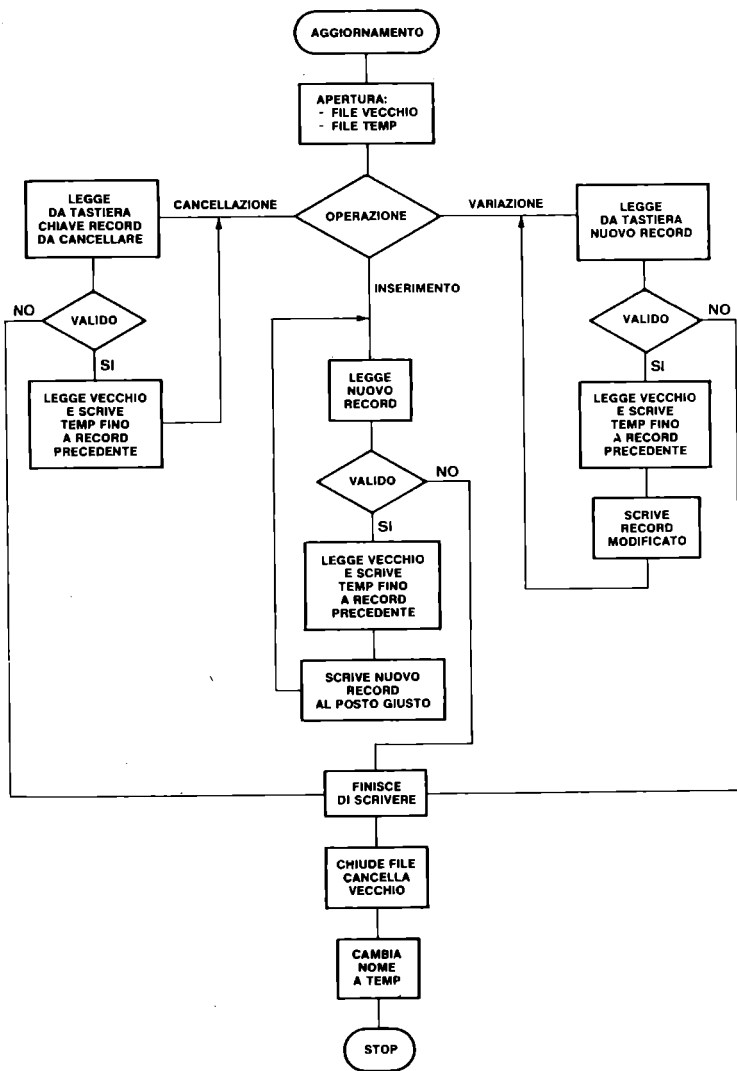



Figura 6.3 Diagramma a blocchi dell'aggiornamento

```
1 REM SEQDISCO
3 REM CREAZIONE E GESTIONE FILE SEQUENZIALE
5 REM ***
```

```
7 REM PARAMETRI DI STAMPA-APERTURA CANALE 15
9 NF=4:PR=4:OPEN15,8,15:GOSUB397
11 REM ***
13 REM COSTANTI E VARIABILI
15 NC=5:CH$=CHR$(13):SP$="    "
17 DIMI$(NC),I1$(NC),D$(NC)
19 DATA "CAMPO1 ","CAMPO2 ","CAMPO3 "
21 DATA "CAMPO4 ","CAMPO5 "
23 FORK=1TONC:READD$(K):NEXTK
25 REM ***
27 REM SCELTA OPERAZIONE
29 PRINT"♥QQCREAZIONE FILE  S/N ";:INPUTR$
31 IFR$<>"S"THEN71
33 REM ***
35 REM CREAZIONE FILE
37 GOSUB297:GOSUB281:GOSUB303
39 PRINT#15,"I0":REM INIZIALIZZAZIONE
41 REM ***
43 REM APERTURA FILE PER SCRIVERE-CANALE 2
45 OPEN2,8,2,"0:"+NF$+",S,W":GOSUB397
47 LC$="":LN$="":REM VECCHI CAMPI 1 E 2
49 GOSUB233:REM RICHIESTA DATI
51 IFSW=0THEN61
53 CLOSE2:CLOSE15
55 PRINT"♥FINITO CARICAMENTO    FILE":STOP
57 REM ***
59 REM CONTROLLO ORDINE RECORD
61 IFI$(1)>LC$THEN67
63 IFI$(1)=LC$THENIFI$(2)>LN$THEN67
65 GOSUB289:GOTO49
67 LC$=I$(1):LN$=I$(2)
69 GOSUB267:GOTO49
71 REM ***
73 REM STAMPA FILE
75 PRINT"♥QQSTAMPA FILE S/N";:INPUTR$
77 IFR$<>"S"THEN103
79 GOSUB297:GOSUB281
81 PRINT#15,"I0":REM INIZIALIZZAZIONE
83 GOSUB303:GOSUB309
85 OPENNF,PR:PRINT#NF,"LISTA ARCHIVIO ";NF$
87 PRINT#NF
89 REM ***
91 REM LETTURA E STAMPA RECORD
93 GOSUB259:PRINT#NF,I$(1);SP$;I$(2)
95 FORK=3TONC:PRINT#NF,I$(K);SP$;:NEXTK
97 PRINT#NF:PRINT#NF:IFFS<>64THEN91
99 CLOSE3:CLOSENF:CLOSE15
101 PRINT"♥FINITO LISTA":STOP
103 REM ***
105 REM AGGIORNAMENTO FILE
107 FF=0:PRINT"♥AGGIORNAMENTO FILE"
109 GOSUB297:GOSUB281
111 PRINT#15,"I0":REM INIZIALIZZAZIONE
```

```
113 GOSUB303:GOSUB309
115 REM ***
117 REM APERTURA FILE TEMPORANEO PER SCRIVERE
119 OPEN2,8,2,"0:TEMP,S,W":GOSUB397
121 LC$="":LN$=""
123 PRINT"[CLR][DOWN][DOWN]TIPO DI VARIAZIONE:"
125 PRINT"1=INSERIMENTO":PRINT"2=VARIAZIONE"
127 PRINT"3=CANCELLAZIONE"
129 INPUTR:IFR<1ORR>3THEN129
131 FL=0:REM 1 SE ULT. REC. LETTO DA SCRIVERE
133 FF=0:REM 1 SE FINITO FILE INPUT
135 ONRGOTO137,185,209:REM SCELTA OPERAZIONE
137 REM ***
139 REM INSERIMENTO RECORD
141 GOSUB233:IFSW=0THEN155
143 GOSUB359:CLOSE3:CLOSE2
145 PRINT"[DOWN]FINITO AGGIORNAMENTO"
147 REM ***
149 REM SISTEMAZIONE FILE SU DISCO
151 PRINT#15,"S0:"+NF$
153 PRINT#15,"R0:"+NF$+"=TEMP":CLOSE15:STOP
155 IFI$(1)>LC$THEN175:REM INSERISCE
157 REM ***
159 REM DATI IN DISORDINE
161 IFI$(1)<LC$THENGOSUB289:GOTO137
163 REM ***
165 REM PRIMO CAMPO UGUALE
167 IFI$(2)>LN$THEN175
169 IFI$(2)<LN$THENGOSUB289:GOTO137
171 GOSUB275:GOTO137
173 REM ***
175 REM DATI IN ORDINE
177 LC$=I$(1):LN$=I$(2):GOSUB317
179 IFI$(1)<>I1$(1)THEN183
181 IFI$(2)=I1$(2)THENGOSUB275:GOTO137
183 GOSUB267:GOTO137
185 REM ***
187 REM MODIFICA RECORD
189 GOSUB233:IFSW=0THEN193
191 GOTO143
193 IFI$(1)>LC$THEN199
195 IFI$(1)=LC$THENIFI$(2)>LN$THEN199
197 GOSUB289:GOTO185
199 LC$=I$(1):LN$=I$(2):GOSUB373
201 IFFL=1ANDFF=0THEN185
203 IFFL=1ANDFF=1THENFL=0:GOTO143
205 GOSUB267:IFFF=1THEN143
207 GOTO185
209 REM ***
211 REM CANCELLAZIONE RECORD
213 PRINTD$(1);:INPUTI$(1):IFI$(1)="*"THEN143
215 PRINTD$(2);:INPUTI$(2):IFI$(1)>LC$THEN221
217 IFI$(1)=LC$THENIFI$(2)>LN$THEN221
```

```
219 GOSUB289:GOTO209
221 LC$=I$(1):LN$=I$(2):GOSUB373
223 IFFF=1ANDFL=0THEN143
225 GOTO209
227 REM ***
229 REM SOTTOPROGRAMMI
231 REM ***
233 REM RICHIESTA DATI
235 SW=0:REM 1 SE FINITI DATI
237 PRINT"▣ 1 ";D$(1);:INPUTI$(1)
239 IFI$(1)="*"THENSW=1:RETURN
241 FORK=2TONC:I$(K)=CHR$(160):NEXTK
243 FORK=2TONC:PRINTK;D$(K);:INPUTI$(K):NEXTK
245 PRINT"▣CONFERMI S/N";:INPUTR$
247 IFR$="S"THENRETURN
249 PRINT"QUALE CAMPO ";:INPUT R
251 IFR<1ORR>NCTHENPRINT"□";:GOTO249
253 PRINTD$(R);:I$(R)=CHR$(160):INPUTI$(R)
255 PRINT"▣";
257 FORK=1TONC:PRINTK;D$(K);I$(K):NEXTK:GOTO245
259 REM LETTURA RECORD DA DISCO
261 FORK=1TONC:INPUT#3,I$(K):FS=ST
263 GOSUB397:NEXTK:RETURN
265 REM ***
267 REM SCRITTURA NUOVO RECORD
269 FORK=1TONC:PRINT#2,I$(K);CH$;
271 GOSUB397:NEXTK:RETURN
273 REM ***
275 REM MESSAGGIO CAMPI 1 E 2 UGUALI
277 PRINT"DATI CAMPI 1 E 2 UGUALI":GOSUB281:RETU
RN
279 REM ***
281 REM ATTESA TASTO
283 GETA$:IFA$=""THEN281
285 RETURN
287 REM ***
289 REM MESSAGGIO FUORI ORDINE
291 PRINT"▣DATI FUORI ORDINE":GOSUB281
293 RETURN
295 REM ***
297 REM RICHIESTA DISCO DATI
299 PRINT"▣MONTA DISCO DATI":RETURN
301 REM ***
303 REM RICHIESTA NOME FILE
305 INPUT"NOME FILE ";NF$:RETURN
307 REM ***
309 REM APERTURA FILE PER LEGGERE-CANALE 3
311 OPEN3,8,3,"0:"+NF$+",S,R":GOSUB397
313 RETURN
315 REM ***
317 REM LEGGE RECORD E SCRIVE FINO AL
319 REM RECORD PRECEDENTE SU TEMP
321 IFFF=1ANDFL=0THENRETURN
```

```
323 IFFL<>0THEN329
325 GOSUB343
327 IFFS=64THENFF=1
329 IFI1$(1)<I$(1)THEN335
331 IFI1$(1)=I$(1)THENIFI1$(2)<I$(2)THEN335
333 FL=1:RETURN
335 GOSUB351:FL=0
337 IFFF=1THENRETURN
339 GOTO325
341 REM ***
343 REM LETTURA RECORD
345 FORK=1TONC:INPUT#3,I1$(K):FS=ST
347 GOSUB397:NEXTK:RETURN
349 REM ***
351 REM SCRITTURA RECORD
353 FORK=1TONC:PRINT#2,I1$(K)
355 GOSUB397:NEXTK:RETURN
357 REM ***
359 REM SCRIVE FILE TEMP FINO ALLA FINE
361 REM DEL FILE DI INPUT
363 IFFF=1ANDFL=0THENRETURN
365 IFFL<>0THENFL=0:GOSUB351:GOTO359
367 GOSUB343:GOSUB351:IFFS=64THENFF=1:RETURN
369 GOTO367
371 REM ***
373 REM COPIA FILE FINO AL RECORD CERCATO
375 IFFF=1ANDFL=0THEN388
377 IFFL<>0THEN381
379 GOSUB343:IFFS=64THENFF=1
381 IFI1$(1)<I$(1)THEN389
383 IFI1$(1)=I$(1)THENIFI1$(2)<I$(2)THEN389
385 IFI1$(1)=I$(1)ANDI1$(2)=I$(2)THENFL=0:RETURN
387 FL=1
388 PRINT"NON TROVATO":GOSUB281:RETURN
389 GOSUB351
391 IFFF=1THEN388
393 GOTO379
395 REM ***
397 REM ROUTINE ERRORE
399 INPUT#15,EN,EM$,ET,ES
401 IFEN=0THEN RETURN
403 PRINT"ERRORE DISCO"
405 PRINTEN,EM$,ET,ES:STOP
```

ELENCO VARIABILI

.NF, 4, numero logico file di stampa.
.PR, 4, dn della periferica di stampa.
.NC, 5, numero campi del record.
.CH$, CHR$(13), carattere separatore RETURN.
.SP$, 4 spazi.

.I$(NC), vettore per leggere i dati dalla tastiera.
.I1$(NC), vettore per leggere da disco il record.
.D$(NC), vettore per le descrizioni dei campi, contenute nelle linee DATA.
.R$, variabile per risposte.
.LC$, primo campo vecchio.
.LN$, secondo campo vecchio.
.FS, memorizzazione parola di stato ST.
.EN, EM$, ET, ES, variabili per controllo errori.
.SW, switch per controllare l'ingresso dei dati: 0 per si dati, 1 per finiti i dati.
.FF, switch per fine file di input, 1 per finito.
.FL, switch per utilizzo ultimo record letto: 0 utilizzato, 1 non utilizzato.
.R, K, variabili per il controllo dei cicli.

Per adattare il programma alle tue esigenze puoi fare le seguenti modifiche:
.PR=3 per listare sul video.
.NC per passare da 5 al numero di campi desiderato.
.linee DATA 19/21, per modificare le diciture dei campi.
.linee da 85 a 103 per ottenere un diverso formato di stampa.

COMMENTO A SEQDISCO
.1/23: definizione costanti e variabili, apertura canale 15, che viene chiuso alla fine di ogni fase.
.25/55: creazione file ex-novo.
.57/69: controllo ordine record nella fase di creazione ex-novo.
.71/103: stampa file.
.105/121: inizio fase di aggiornamento, apertura file temporaneo.
.123/135: scelta tipo aggiornamento, azzeramento switch.
.137/145: fase inserimento nuovi record.
.147/155: sistemazione file temporaneo per fine aggiornamento.
.157/171: controllo ordine dei record e uguaglianza primi 2 campi nella fase di inserimento.
.173/183: inserimento se dati in ordine.
.185/207: fase aggiornamento record per modifiche.
.209/225: fase cancellazione record.
.227/263: inizio sezione per i sottoprogrammi. Richiesta dati da tastiera.
.265/271: scrittura nuovo record.
.273/277: messaggio per campi chiave uguali.
.279/285: attesa tasto per proseguire.
.287/293: messaggio record fuori ordine.
.295/299: richiesta disco dati.
.301/305: richiesta nome file.

.307/313: apertura file per leggere.
.315/339: avanzamento e copiatura file fino al record precedente, per la fase di modifica record.
.341/347: lettura record da disco.
.349/355: scrittura record su TEMP.
.357/369: finisce di ricopiare file su TEMP.
.371/393: copia file fino al record cercato, per la fase di cancellazione.
.395/405: routine di controllo errore disco.

6.6 FILE RANDOM DI DATI

Questa denominazione di file e' abbastanza impropria, ma ormai in uso; per questa ragione anche noi chiamiamo random questi file, che in realta' sono ad accesso diretto. Essi sono gestiti con il gruppo dei comandi DOS per l'accesso diretto, che non comportano una registrazione nella directory del disco. La cosa non e' consigliabile, dal momento che su un floppy che contenga file random non si possono eseguire i comandi VALIDATE e COLLECT che controllano e sistemano la BAM e la directory. Noi per completare l'argomento presentiamo un programma che gestisce in modo RANDOM un archivio di dati, in modo pericoloso, cioe' senza registrazione nella BAM; e un secondo programma che gestisce un file in modo RANDOM-USER, con l'accorgimento di preparare preventivamente il floppy in modo che il file sia anche registrato nella directory. In questo secondo caso sono eseguibili sul floppy anche i comandi VALIDATE e COLLECT.
Nell'organizzazione diretta il file viene gestito a livello di record fisico, cioe' di settore, conoscendone l'indirizzo t,s.
Il record logico deve essere definito con tutti i suoi campi e adattato alla struttura fisica. Senza addentrarsi in tecniche di programmazione piuttosto sofisticate, il modo migliore per gestire questo tipo di file e' di definire un record logico di lunghezza fissa, formato da un numero fisso di campi di lunghezza fissa, in modo tale che risulti di lunghezza multipla o sottomultipla di un settore, considerato di 254 caratteri. Preferiamo usare solo 254 caratteri in un settore per poter mantenere l'uso dei primi 2 byte per il concatenamento tra i settori, come per gli altri tipi di file.
Dobbiamo ora affrontare un altro argomento relativo all'organizzazione dei file, cioe' la possibilita' di reperire rapidamente il record voluto. Nell'esempio relativo ai file sequenziali abbiamo preparato il file mantenendo i record in ordine in base a due campi chiave, pero' la ricerca poteva avvenire solo leggendo i record in sequenza. In questo caso possiamo accedere al record desiderato, se conosciamo l'indirizzo t,s. Come facciamo a sapere, per esempio in un archivio relativo a una biblioteca, che il libro ARITMETICA PRATICA dell'autore PIPPO si trova in un determinato settore? Potremmo avere una lista su carta che

ci da' queste informazioni, cioe' un indice da consultare manualmente prima di una ricerca. Ma, dal momento che abbiamo un calcolatore, e' meglio organizzarci per fargli svolgere tutto il lavoro in modo automatico.
In conseguenza quando creiamo un file ad accesso diretto, fisico per i file random, logico per i file relativi che trattiamo nel prossimo paragrafo, dobbiamo creare anche un indice sequenziale, organizzato in base a uno o piu' campi di ordinamento, che ci fornisca gli indirizzi, fisici o logici, necessari per reperire un record.
Il file INDICE, che e' un normale file sequenziale, deve essere abbastanza corto se possibile. L'altro file, quello diretto, si chiama PRINCIPALE. La situazione ideale e' che sia possibile caricare in memoria il file indice prima di iniziare l'elaborazione del file principale; in tale modo le operazioni di ricerca nell'indice risultano molto veloci. Se questo non e' possibile, il file indice puo' venir letto a blocchi, andando a leggere i blocchi successivi quando servono. Comunque la lettura del file indice risulta piu' veloce di quanto sarebbe il trattamento di un file principale, con tanti campi, di tipo sequenziale.

Nel Paragrafo 6.2, nella sezione GESTIONE DIRETTA, sono riportati i comandi per il DOS. Il trattamento dei file random impegna 2 canali, il canale 15, per i comandi, e un canale, da 2 a 14 per i dati. Nel buffer dei dati agiscono le normali istruzioni PRINT#, INPUT# e GET#.
Nel seguito riportiamo due esempi di programmi per gestire un archivio random normale e uno organizzato in modo da proteggere le registrazioni. Nei commenti trattiamo completamente l'argomento.
Ti facciamo notare che i file random, dopo l'apertura, possono essere letti e scritti, mescolando le due operazioni.

ARCHIVIO RANDOM

Abbiamo preparato il programma ARCHIRANDOM per gestire un file RANDOM con indice. Le funzioni del programma sono le seguenti:
.1) crea sul floppy un archivio RANDOM, che non ha un nome registrato nella directory, e il file indice INDI1, usando come campi chiave i primi due campi del record logico, e ordinando l'indice in ordine crescente in base ai campi chiave.
.2) lista tutto l'archivio in ordine secondo l'indice INDI1.
.3) aggiorna l'archivio operando:
.3a) aggiunta di nuovi record,
.3b) modifica di record esistenti,
.3c) cancellazione di record.
Gli aggiornamenti avvengono sul file principale e sull'indice INDI1.

.4) crea un indice secondario INDI2 in base a un qualunque campo del record, escluso il primo, in ordine decrescente del campo chiave.
.5) lista tutto l'archivio in ordine secondo INDI2.

Per gestire veramente un archivio di dati sono necessarie anche altre funzioni, come ricerca e stampa di un record o di un gruppo di record, estrazione di record con determinate caratteristiche, ecc.. Partendo dal nostro esempio, non ti sara' difficile sviluppare in modo simile altre parti del programma. Il nostro scopo e' di mostrarti la problematica della gestione degli archivi di dati e alcuni esempi completi dell'uso delle istruzioni del DOS per la gestione diretta dei file.

Il programma e' stato sviluppato con la tecnica dei sottoprogrammi e in modo che, con poche modifiche, puo' essere adattato ad altre situazioni.
Abbiamo organizzato un record logico formato da 14 campi di lunghezza fissa, realizzando un archivio anagrafico. Elenchiamo nell'ordine di registrazione le descrizioni dei campi e la loro lunghezza in caratteri:

1) cognome	20		8) data nascita	8
2) nome	15		9) titolo studio	20
3) indirizzo	30		10) occup. attuale	20
4) CAP/citta'	25		11) occup. preced.	20
5) provincia	11		12) stato civile	1
6) telefono	10		13) nota 1	20
7) luogo nascita	20		14) nota 2	20

alla lunghezza di ogni campo devi aggiungere un carattere di fine campo, che per noi e' CHR$(13) (RETURN). Avremmo potuto usare anche CHR$(44) (virgola), ma con il carattere RETURN siamo piu' liberi di leggere con ogni INPUT# i campi desiderati. La somma delle lunghezze dei campi piu' il carattere separatore da' 254, cioe' un settore non usando i primi due caratteri. Abbiamo quindi coincidenza tra il record logico e il record fisico. I dati sono registrati nel settore partendo dal terzo carattere, cioe' posizionando il puntatore interno a 2.
Per il file indice INDI1 ogni record e' formato dai seguenti campi:

. chiave: cognome+nome, di 35 caratteri, completando con spazi ogni campo se piu' corto della sua lunghezza,
. indirizzo traccia, t, numero intero,
. indirizzo settore, s, numero intero.

Per il file INDI2, che viene creato quando si esegue il passo 4, abbiamo 3 campi per record, come per INDI1, ma le dimensioni del primo campo dipendono dal campo scelto come chiave. Dal momento

che un campo chiave puo' anche essere vuoto, se esso ha lunghezza 0, viene sostituito in INDI2 dal carattere CHR$(160). In conseguenza nell'ordinamento decrescente di INDI2 i campi chiave vuoti risultano i primi.

Dobbiamo fare alcune considerazioni sull'occupazione di memoria. Infatti l'indice INDI1 viene creato e gestito in memoria e poi scritto su disco, quindi richiede spazio per i 3 vettori nei quali viene memorizzato. Non ci sembra utile fare dei conti precisi per il nostro archivio, dal momento che tu probabilmente personalizzerai il programma secondo le tue esigenze. Per fare i conti, dopo aver caricato in memoria il programma, devi procedere cosi':
.leggere il puntatore a inizio variabili, nei byte 45 e 46, e calcolarne il valore;
.leggere il puntatore di fine zona BASIC, nei byte 55 e 56, e calcolarne il valore;
.calcolare la differenza tra il secondo e il primo valore, che rappresenta il numero di byte a disposizione per tutte le variabili (tale numero lo ottieni anche eseguendo CLR:PRINT FRE(0));
.calcolare approssimativamente l'occupazione di memoria per ogni record di INDI1 e quindi ricavare il numero di elementi che puo' avere ognuno dei 3 vettori. A questo punto hai fatto i conti con la memoria del calcolatore.
Per il dischetto devi tener presente che ogni record del file random occupa un settore, calcolare l'occupazione di ogni record di INDI1 e di INDI2 in modo approssimato, dedurre il numero di record registrabili e controllare se tale numero si accorda con il calcolo fatto per la memoria del calcolatore.
E' importante fare bene questi conti per non avere spiacevoli sorprese. In caso un archivio puo' essere suddiviso in diverse parti e tenuto su piu' floppy.

Il programma lavora usando un dischetto apposito per i dati; su questo nel settore 0 della traccia 1 sono memorizzati i dati generali del file, e precisamente:
. data di aggiornamento,
. numero dei record presenti,
. indirizzo del blocco, traccia e settore, dell'ultimo record registrato.
I record del file random sono registrati a partire dalla traccia 1, settore 1.
All'inizio viene presentato un menu' di scelta della funzione. Il programma procede con domande semplici nelle diverse situazioni. Per uscire dalla richiesta dati devi rispondere con il carattere dollaro "$" al cognome. Quando compare un messaggio senza richiesta di risposta per proseguire devi premere un tasto.
Per uscire dal menu' senza operare devi scegliere la funzione 9 (FINE).

```
1 REM ARCHIRANDOM
3 REM ***********
5 REM DEFINIZIONE COSTANTI E VARIABILI
7 REM ********************************
9 NC=14:REM NUMERO CAMPI RECORD
11 S1$="DATA ULTIMO AGG.      "
13 S2$="FINITO SPAZIO ASSEGNATO"
15 SP$="":FORK=1TO79:SP$=SP$+" ":NEXTK
17 CH$=CHR$(13):LIM$=CHR$(99)+CHR$(99)+CHR$(99)
19 DIMD$(NC):DIMY$(NC):DIML(NC)
21 DATA"COGNOME:","NOME    :","INDIR. :"
23 DATA"CITTA' :","PROV.   :","TELEF. :"
25 DATA"L.NASC.:","D.NASC.:","T.STUD.:"
27 DATA"OCC.AT.:","OCC.PR.:","ST.CIV.:"
29 DATA"NOTA 1  :","NOTA 2  :"
31 DATA20,15,30,25,11,10,20,8,20,20,20,1,20,20
33 FORK=1TONC:READD$(K):NEXTK
35 FORK=1TONC:READL(K):NEXTK
37 REM PRESENTAZIONE MENU' E SCELTE
39 REM *****************************
41 OPEN15,8,15
43 PRINT"♥QQ";TAB(10);"GESTIONE ARCHIVIO":PRINT
45 PRINTTAB(10)"1=INIZIO EX-NOVO"
47 PRINTTAB(10)"2=AGGIORNAMENTO"
49 PRINTTAB(10)"3=LISTA PRINCIPALE"
51 PRINTTAB(10)"4=CREAZ.IND.SEC."
53 PRINTTAB(10)"5=LISTA SECONDARIA"
55 PRINTTAB(10)"9=FINE"
57 INPUT"COSA SCEGLI ";X
59 IFX<1ORX>5ANDX<>9THENPRINT"□";:GOTO57
61 PRINT:GOSUB225
63 R$="":PRINTTAB(10)"MONTA DISCO DATI"
65 GETR$:IFR$=""THEN65
67 IFX=1THEN77
69 PRINT#15,"I"
71 GOSUB185:N=K:PRINT"PRESENTI ";K;" RECORD"
73 PRINTS1$;G1$;"/";M1$;"/";A1$
75 IFX<>2THEN83
77 REM INIZIO EX-NOVO ARCHIVIO
79 REM ***********************
81 PRINT"      QUANTI RECORD ";:INPUT N
83 DIMC$(N),T%(N),S%(N)
85 IFX<>1THENGOSUB201
87 IFX=9THEN507
89 ONXGOTO251,303,435,467,489
91 STOP
93 REM INGRESSO DATI
95 REM *************
97 PRINT"♥INGRESSO DATI"
99 FORJ=1TONC:Y$(J)="":NEXTJ
101 PRINT"PER USCIRE $ PER COGNOME";
103 PRINT"▌SE MANCANO DATIPREMI SOLO RETURN"
105 PRINTD$(1);:INPUTY$(1)
```

```
107 IFY$(1)="$"THENW=1:RETURN
109 FORJ=2TONC:PRINTD$(J);:INPUTY$(J):NEXTJ
111 REM SISTEMA LUNGHEZZA DATI
113 REM **********************
115 FORJ=1TONC:Y$(J)=LEFT$(Y$(J)+SP$,L(J))
117 NEXTJ:GOSUB379:RETURN
119 REM SCRITTURA INDICE PRINCIPALE
121 REM ***************************
123 PRINT#15,"S0:INDI1":PRINT#15,"I"
125 OPEN10,8,10,"INDI1,S,W":GOSUB213
127 FORJ=1TOK
129 PRINT#10,C$(J);CH$;T%(J);CH$;S%(J);CH$;
131 GOSUB213:NEXTJ
133 CLOSE10:RETURN
135 REM SCRITTURA NEL BUFFER
137 REM ********************
139 FORJ=1TONC:Y$(J)=LEFT$(Y$(J)+SP$,L(J))
141 PRINT#11,Y$(J);CH$;:GOSUB213
143 NEXTJ:RETURN
145 REM ALLOCA TRACCIA E SETTORE
147 REM ************************
149 PRINT#15,"B-A:"0;T;S
151 INPUT#15,EN,EM$,ET,ES
153 IFEN=0THENRETURN
155 IFEN<>65THEN221
157 IFET=18THENT=19:S=0:GOTO149
159 T=ET:S=ES:GOTO149
161 REM PUNTATORE NEL BUFFER
163 REM ********************
165 PRINT#15,"B-P:"11;2:GOSUB213:RETURN
167 REM SCRITTURA RECORD
169 REM ****************
171 PRINT#15,"U2:"11;0;T;S:GOSUB213:RETURN
173 REM LETTURA RECORD
175 REM **************
177 PRINT#15,"I":OPEN11,8,11,"#":GOSUB213
179 PRINT#15,"U1:"11;0;T;S:GOSUB213:GOSUB165
181 FORJ=1TONC:INPUT#11,Y$(J):GOSUB213
183 NEXTJ:CLOSE11:RETURN
185 REM LETT. DATI DISCO
187 REM ****************
189 OPEN11,8,11,"#":GOSUB213
191 PRINT#15,"U1:"11;0;1;0:GOSUB213
193 GOSUB161:INPUT#11,R$,K,T,S:GOSUB213
195 G1$=LEFT$(R$,2):M1$=MID$(R$,3,2)
197 A1$=RIGHT$(R$,2):CLOSE11
199 TT=T:SS=S:RETURN
201 REM LETT. INDICE
203 REM ************
205 OPEN10,8,10,"INDI1,S,R":GOSUB213
207 FORJ=1TOK
209 INPUT#10,C$(J),T%(J),S%(J):GOSUB213
211 NEXTJ:CLOSE10:RETURN
```

```
213 REM ROUTINE ERRORE
215 REM **************
217 INPUT#15,EN,EM$,ET,ES
219 IFEN=0THENRETURN
221 PRINT"QERRORE DISCO"
223 PRINTEN,EM$,ET,ES:CLOSE15:STOP
225 REM DATA DISCO
227 REM **********
229 PRINT"DATA PER DISCO"
231 INPUT"  GG,MM,AA███████████";G$,M$,A$:RETURN
233 REM SCRITT. IND. SEC.
235 REM *****************
237 PRINT#15,"S:INDI2":PRINT#15,"I"
239 OPEN10,8,10,"INDI2,S,W"
241 FORJ=1TOK
243 PRINT#10,C$(J);CH$;T%(J);CH$;S%(J);CH$;
245 GOSUB213:NEXTJ:CLOSE10:RETURN
247 GETR$:IFR$=""THEN247
249 RETURN
251 REM INIZIALIZZAZIONE DISCO
253 REM **********************
255 PRINT"NOME DISCO";:INPUTN$:T=1:S=0
257 REM DATI DISCO SETTORE 1,0
259 REM **********************
261 PRINT#15,"N0:"+N$+",99"
263 CLOSE15:OPEN15,8,15:PRINT#15,"I"
265 REM DATA AGG. E NUM. RECORD
267 REM ***********************
269 OPEN11,8,11,"#":GOSUB213:GOSUB149:GOSUB165
271 K=0
273 PRINT#11,G$M$A$CH$;K;CH$;1;CH$;1;CH$;
275 REM PRIMO SETTORE DATI 1,1
277 REM **********************
279 GOSUB213
281 PRINT#15,"U2:"11;0;T;S:GOSUB213
283 K=1:W=0:T=1:S=1
285 GOSUB93
287 IFW=1THENK=K-1:CLOSE11:GOTO507
289 GOSUB145:GOSUB161:GOSUB135
291 REM AGGIORNAMENTO INDICE
293 REM ********************
295 C$(K)=Y$(1)+Y$(2):T%(K)=T:S%(K)=S
297 GOSUB167:K=K+1
299 IFK>NTHENK=K-1:PRINTS2$:CLOSE11:GOTO507
301 GOTO285
303 REM AGGIORNAMENTO
305 REM *************
307 PRINTTAB(10);"QQQAGGIORNAMENTO ARCHIVIO"
309 PRINT:PRINTTAB(10);"1=CORREZIONE"
311 PRINTTAB(10);"2=AGGIUNTA ELEM."
313 PRINTTAB(10);"3=CANCELL.ELEM."
315 PRINTTAB(10);"9=FINE"
317 INPUT"COSA SCEGLI ";X:IFX=9THEN507
```

```
319 IFX<1ORX>3THEN307
321 IFX=2THEN401
323 IFX=3THEN417
325 GOTO359
327 REM RICERCA RECORD
329 REM **************
331 PRINT"";D$(1);:Y$(1)="":INPUTY$(1):W=0
333 IFY$(1)="$"THENW=1:RETURN
335 PRINTD$(2);:Y$(2)="":INPUTY$(2)
337 Y$(1)=LEFT$(Y$(1)+SP$,L(1))
339 Y$(2)=LEFT$(Y$(2)+SP$,L(2))
341 INPUT"QUALE OCCORRENZA ";X
343 IFX<=0THENPRINT"";:GOTO341
345 FORJ=1TOK:IFC$(J)=Y$(1)+Y$(2)THEN353
347 NEXTJ
349 J=J-1:PRINT"NON TROVATO ";Y$(1);"  ";Y$(2)
351 GOSUB247:GOTO327
353 IFX<>1THENX=X-1:GOTO347
355 T=T%(J):S=S%(J)
357 I=J:J=K:NEXTJ:RETURN
359 REM CORREZIONE
361 REM **********
363 GOSUB327:IFW=1THEN307
365 GOSUB173
367 REM VA A MODIFICA DATI
369 REM ******************
371 GOSUB379
373 PRINT#15,"I":OPEN11,8,11,"#":GOSUB161:GOSUB1
35
375 C$(I)=Y$(1)+Y$(2):T%(I)=T:S%(I)=S
377 GOSUB167:CLOSE11:GOTO325
379 REM CONTROLLO DATI E MODIFICHE
381 REM **************************
383 PRINT"CONTROLLO DATI E MODIFICHE"
385 FORJ=1TONC:PRINTJ;" ";D$(J);" ";Y$(J):NEXTJ
387 INPUT"TUTTO BENE S/N ";X$:IFX$="S"THENRETUR
N
389 PRINT"QUALE CAMPO (0 USCITA)";:INPUTX
391 IFX=0THENRETURN
393 IFX>NCORX<1THENPRINT"":GOTO387
395 Y$(X)=""
397 INPUTY$(X):Y$(X)=LEFT$(Y$(X)+SP$,L(X))
399 GOTO383
401 REM AGGIUNTA
403 REM ********
405 OPEN11,8,11,"#":GOSUB213
407 PRINT"AGGIUNTA NUOVI RECORD"
409 PRINT"PRESENTI ";K;" RECORD"
411 PRINT"PUOI AGGIUNGERE ";N-K;" RECORD"
413 GOSUB247
415 T=TT:S=SS:W=0:K=K+1:GOTO285
417 REM CANCELLAZIONE
```

```
419 REM *************
421 M=0
423 GOSUB327:IFW=1THEN431
425 X=L(1)+L(2):C$(I)=LEFT$(LIM$+SP$+SP$,X)
427 M=M+1
429 PRINT#15,"B-F:"0;T;S:GOSUB213:GOTO423
431 REM SIST. INDICE
432 REM ************
433 GOSUB533:K=K-M:GOTO517
435 REM LISTA FILE
437 REM **********
439 T$="LISTA PER INDICE PRIM."
441 PRINT"ACCENDI STAMPANTE      PREMI UN TASTO"
443 GETR$:IFR$=""THEN443
445 PRINT"{CLR}";T$:OPEN4,4
447 PRINT#4:PRINT#4:PRINT#4,T$
449 FORJ=1TO10:PRINT#4:NEXTJ
451 L=2:FORI=1TOK:T=T%(I):S=S%(I):GOSUB173
453 PRINT#4,Y$(1);"  ";Y$(2)
455 PRINT#4,Y$(3);"  ";Y$(4);"  ";Y$(5)
457 FORM=6TONC:PRINT#4,D$(M);Y$(M):NEXTM
459 PRINT#4:PRINT#4
461 IFL=5THENPRINT#4:L=0
463 L=L+1:NEXTI
465 CLOSE4:GOTO527
467 REM CREAZIONE INDICE SECONDARIO
469 REM ***************************
471 PRINT"{CLR}INDICE SECONDARIO"
473 INPUT"{DOWN}QUALE CAMPO ";X$
475 X=VAL(X$):IFX<2ORX>NCTHENGOTO473
477 GOSUB119
479 FORI=1TOK:T=T%(I):S=S%(I):GOSUB173
480 IFLEN(Y$(X))=0THENY$(X)=CHR$(160)
481 C$(I)=Y$(X):NEXTI
483 GOSUB233:GOSUB201
485 PRINT"FINITO IND. SEC."
487 GOTO527
489 REM LISTA PER INDICE SECONDARIO
491 REM ***************************
493 PRINT"{CLR}LISTA IND. SEC."
495 OPEN10,8,10,"INDI2,S,R":GOSUB213
497 FORJ=1TOK:INPUT#10,C$(J),T%(J),S%(J)
499 GOSUB213:NEXTJ
501 CLOSE10:GOSUB561:GOSUB233
503 T$="LISTA IND. SEC. "
505 GOTO441
507 REM CHIUSURA
509 REM ********
511 PRINT"CHIUSURA ARCHIVIO"
513 PRINT"IND. PRINC. DA ORDIN. S/N ":INPUTR$
515 IFR$="S"THENGOSUB533
517 GOSUB119:OPEN11,8,11,"#":GOSUB213
519 PRINT#15,"B-P:"11,2
```

```
521 PRINT#11,G$M$A$CH$;K;CH$;T;CH$;S;CH$;
523 GOSUB213
525 PRINT#15,"U2:"11;0;1;0:GOSUB213:CLOSE11
527 PRINT"FINITO AGGIORNAMENTO"
529 PRINT"SONO PRESENTI ";K;" RECORD"
531 CLOSE15:STOP
533 REM ORDINAMENTO CRESCENTE
535 REM ********************
537 L=K-1
539 W=0
541 FORJ=1TOL
543 IFC$(J)<=C$(J+1)THEN553
545 R$=C$(J):C$(J)=C$(J+1):C$(J+1)=R$
547 X=T%(J):T%(J)=T%(J+1):T%(J+1)=X
549 X=S%(J):S%(J)=S%(J+1):S%(J+1)=X
551 W=1
553 NEXTJ
555 IFW=0THENRETURN
557 IFL=1THENRETURN
559 L=L-1:GOTO539
561 REM ORDINAMENTO DECRESCENTE
563 REM **********************
565 L=K-1
567 W=0
569 FORJ=1TOL
571 IFC$(J)>=C$(J+1)THEN581
573 R$=C$(J):C$(J)=C$(J+1):C$(J+1)=R$
575 X=T%(J):T%(J)=T%(J+1):T%(J+1)=X
577 X=S%(J):S%(J)=S%(J+1):S%(J+1)=X
579 W=1
581 NEXTJ
583 IFW=0THENRETURN
585 IFL=1THENRETURN
587 L=L-1:GOTO567
```

ELENCO VARIABILI

.NC, numero campi del record logico, 14 per noi.

.S1$,S2$,T$, stringhe descrittive.

.SP$, 79 spazi, tale stringa deve contenere un numero di spazi pari al campo piu' lungo del record, infatti serve per aggiungere spazi ai campi piu' corti o vuoti. Noi abbiamo usato 79, per la compatibilita' con altri calcolatori COMMODORE, ma potrebbe essere anche 88.

.CH$, CHR$(13), per separare i campi con RETURN.

.LIM$, tre caratteri CHR$(99), serve per il campo chiave dei record cancellati.

.D$(NC), descrizioni dei campi del record, da prelevare dalle frasi DATA presenti nelle linee da 21 a 29.
.Y$(NC), campi dati.
.L(NC), lunghezze dei campi dati, da prelevare dalla linea DATA 31.
.K,J,L, variabili di controllo.
.X,R$,X$, variabili per risposte.
.N, numero record.
.G1$,M1$,A1$, data precedente.
.G$,M$,A$, data aggiornamento.
.C$(N), chiavi dell'indice.
.T%(N),S%(N), tracce e settori dell'indice.
.W, switch per lettura dati, 1 se finiti i dati.
.T,S, traccia e settore.
.EN,EM$,ET,ES, variabili per messaggio errore.
.N$, nome disco e archivio.
.TT,SS, indirizzo settore precedente.
.M, numero record cancellati.

Le modifiche per adattare il programma alle tue esigenze riguardano:
.il numero dei campi NC,
.le frasi DATA delle descrizioni e delle lunghezze dei campi.
.la parte di programma che lista i record.
Risulta piu' difficile modificare il fatto che l'indice e' sui primi due campi; puoi ovviare all'inconveniente dimensionando uno dei due campi a lunghezza 1 e lasciandolo vuoto.
Le due routine di ordinamento risultano un po' lente; potrebbero essere sostituite con routine di ordinamento in linguaggio macchina. Esse lavorano sui 3 vettori che contengono l'indice, o principale o secondario.
Il programma accetta record con i primi due campi uguali; in conseguenza nella ricerca chiede anche l'occorrenza del record.

COMMENTO A ARCHIRANDOM
.1/35: definizione costanti e variabili.
.37/91: presentazione menu' e scelte.
.93/117: sottoprogramma ingresso dati e sistemazione lunghezza campi.
.119/133: sottoprogramma scrittura INDI1 su floppy.
.135/143: sottoprogramma scrittura nel buffer.
.145/159: sottoprogramma allocazione traccia e settore.
.161/165: sottoprogramma posizionamento puntatore.
.167/171: sottoprogramma scrittura settore sul floppy.
.173/183: sottoprogramma lettura record.
.185/199: sottoprogramma lettura dati generali disco.
.201/211: sottoprogramma lettura INDI1 in memoria.
.213/223: sottoprogramma routine errore disco.

.225/231: sottoprogramma richiesta data aggiornamento.
.233/249: sottoprogramma scrittura indice INDI2.
.251/301: fase 1 del menu', inizio ex-novo archivio e caricamento record.
.303/433: fase 2 del menu', aggiornamento: modifica, aggiunta e cancellazione. Quando si cancella un record viene liberato il settore occupato.
.435/465: fase 3 del menu', lista archivio.
.467/487: fase 4 del menu', creazione INDI2.
.489/531: fase 5 del menu', lista per indice secondario.
.533/559: sottoprogramma ordinamento crescente indice, con eliminazione dei record cancellati.
.561/587: sottoprogramma ordinamento decrescente indice.

ARCHIVIO RANDOM-USER

Per organizzare un archivio di questo tipo e' necessario eseguire preventivamente un programma che impegni su disco il numero di settori che servono per il file random, creando un file sequenziale di record "dummy" (vuoti), al quale, per distinguerlo dagli altri, possiamo assegnare il tipo USR. Impegnando i settori in questo modo essi vengono concatenati sui primi 2 caratteri e il file di tipo USR viene registrato nella directory. Dobbiamo conservare gli indirizzi di traccia e settore utilizzati dal file USR, e, quando scriviamo i record del file random, dobbiamo andare a scrivere sugli stessi settori, senza eseguire il comando B-A di allocazione. In questo modo le nostre registrazioni non vengono danneggiate dai comandi COLLECT e VALIDATE.

Abbiamo voluto migliorare ulteriormente l'organizzazione del nostro programma, in modo che esso possa essere utilizzato per generare archivi di tipo diverso. Abbiamo stabilito di mantenere la lunghezza del record logico al massimo a 254 caratteri, utilizzando un settore per ogni record logico. Inoltre abbiamo mantenuto le caratteristiche funzionali del programma ARCHIRANDOM.
Per realizzare questo archivio dobbiamo:
.preparare un programma, di nome STRUTTFILE, che mediante un colloquio video/tastiera, ci consente di definire la struttura del record logico dell'archivio, fissando il numero dei campi, il loro nome e la loro lunghezza, il nome del disco e dell'archivio, la data iniziale. STRUTTFILE controlla che la lunghezza dei campi non superi 254. Le caratteristiche definite per l'archivio vengono scritte su un file di tipo sequenziale di nome STRUTTURA; questo file viene letto dal programma di gestione dell'archivio all'inizio dell'elaborazione e gli fornisce le costanti, che per ARCHIRANDOM sono incorporate nel programma stesso.

.preparare un programma, di nome INIZIOFILE, che, ricavando le informazioni necessarie dal file STRUTTURA, crea sul floppy un file sequenziale di tipo USR, riservando tutti i settori che servono per il file random, e inoltre crea a priori il file sequenziale INDI1, lasciando vuote le chiavi, ma scrivendo gli indirizzi di traccia e settore utilizzati.

.preparare un programma, di nome ARANDOMUSER, che gestisce il file random, utilizzando il file STRUTTURA e il file INDI1.

```
1 REM STRUTTFILE
3 REM **********
5 S1$="LA SOMMA DELLE LUNGHEZZE SUPERA 254"
7 S2$="MODIFICA QUALCHE CAMPO"
9 PRINT"♥DEFINIZIONE STRUTTURA RECORDQ"
11 REM CHIEDE NUMERO CAMPI MINORE DI 20
13 REM ********************************
15 INPUT"NOME FILE: ";N$
17 INPUT"QUANTI CAMPI: ";NC
19 IFNC<2ORNC>20THEN9
21 PRINT"♥NOMI E LUNGHEZZE CAMPI"
23 PRINT"OGNI CAMPO MINORE DI 80 CARATT."
25 PRINT"OGNI NOME MASSIMO 10 CARATTERI"
27 DIMD$(NC),L(NC)
29 FORK=1TONC
31 PRINTK;"NOME: ";:INPUTD$(K)
33 INPUT"LUNGHEZZA: ";L(K)
35 NEXTK
37 PRINT"♥CONTROLLO CAMPI"
39 FORK=1TONC
41 IFLEN(D$(K))>10THEND$(K)=LEFT$(D$(K),10)
43 IFL(K)>79THENL(K)=79
45 PRINTK;D$(K);" ";L(K)
47 NEXTK
49 PRINT"QCONFERMI S/N ";:INPUTR$
51 IFR$="S"THEN61
53 INPUT"QUALE CAMPO: ";X
55 IFX<0ORX>NCTHEN53
57 INPUT"CAMPO: ";D$(X)
59 INPUT"LUNGHEZZA: ";L(X):GOTO37
61 REM CALCOLA LUNGHEZZA RECORD
63 REM ************************
65 S=NC:FORK=1TONC
67 S=S+L(K):NEXTK
69 IFS>254THENPRINTS1$:PRINTS2$:GOTO39
71 PRINT"NOME FILE: ";N$
73 PRINT"LUNGHEZZA RECORD: ";S
75 CLOSE15:OPEN15,8,15,"I"
76 PRINT#15,"S:STRUTTURA"
77 OPEN2,8,2,"STRUTTURA,S,W"
79 PRINT#2,N$
81 PRINT#2,NC
```

```
83 FORK=1TONC:PRINT#2,D$(K):NEXTK
85 FORK=1TONC:PRINT#2,L(K):NEXTK
87 CLOSE2
89 PRINT"♥VERIFICA"
91 OPEN2,8,2,"STRUTTURA,S,R"
93 INPUT#2,A$
95 PRINT"NOME FILE: ";A$
97 INPUT#2,X
99 PRINT"NUMERO CAMPI: "X
101 FORK=1TOX:INPUT#2,D$(K):NEXTK
103 FORK=1TOX:INPUT#2,L(K):NEXTK
105 CLOSE2:CLOSE15
107 FORK=1TOX:PRINTK;D$(K);L(K):NEXTK
109 STOP
```

COMMENTO A STRUTTFILE

.15: chiede il nome del file random e lo pone in N$.

.17: chiede il numero NC dei campi e controlla che sia compreso tra 3 e 20. Abbiamo imposto il minimo 2, dato che per il nostro file la chiave di ordinamento lavora sui primi 2 campi. Il limite superiore 20 ci consente di vedere in un solo quadro video i campi.

.21/35: chiede il nome e la lunghezza dei campi, con la limitazione che il nome non superi 10 caratteri e la lunghezza 79 caratteri (sempre per la compatibilita' con altri modelli).

.37/47: controlla i dati e li aggiusta ai limiti se necessario.

.49/59: chiede conferma dei dati e consente di correggerli.

.61/69: controlla la lunghezza del record, aggiungendo alla lunghezza di ogni campo 1 per il CHR$(13) di fine campo. Se non risulta <=254 chiede di modificare la struttura del record.

.71/87: scrive il file STRUTTURA sul dischetto del programma, dopo aver cancellato, se esiste, un precedente file.

.89/109: rilegge il file STRUTTURA per verificare i dati e li ripropone sul video.

```
1 REM INIZIOFILE
3 REM **********
5 REM DIMENSIONA COSTANTI E VARIABILI
7 REM LEGGE FILE STRUTTURA
9 REM ******************************
11 CH$=CHR$(13)
13 S1$=" ":FORK=1TO39:S1$=S1$+" ":NEXTK
15 S2$="         "
17 OPEN15,8,15,"I"
19 OPEN2,8,2,"STRUTTURA,S,R":GOSUB229
21 INPUT#2,N$,NC:GOSUB229
23 N$=LEFT$(N$+S1$,16)
```

```
25 PRINT"DATI FILE STRUTTURA"
27 PRINT"FILE: ";N$;" NUMERO CAMPI: ";NC
29 INPUT"QUANTI RECORD: ";N
31 IFN>100THEN STOP
33 NX$=STR$(N)
35 NX$=RIGHT$(S2$+NX$,6)
37 DIMD$(NC),L(NC),Y$(NC)
39 FORK=1TONC:INPUT#2,D$(K):NEXTK
41 FORK=1TONC:INPUT#2,L(K):NEXTK
43 CLOSE2
45 FORK=1TONC:PRINTD$(K);"   ";L(K):NEXTK
47 SR=NC:FORK=1TONC:SR=SR+L(K):NEXTK
49 PRINT"LUNGHEZZA RECORD: ";SR
51 IFSR>254THENPRINT"NON POSSO PROSEGUIRE":STOP
53 REM CAMPI DATI CON CHR$(99) + SPAZI
55 REM ********************************
57 FORK=1TONC:Y$(K)=CHR$(99):IFL(K)=1THEN63
59 FORJ=1TOL(K)-1
61 Y$(K)=Y$(K)+CHR$(32):NEXTJ
63 NEXTK
65 IFSR=254THEN77
67 REM CREA CAMPO FINALE SE SR<254
69 REM ****************************
71 IFSR=253THENDN$="":GOTO77
73 DN$=CHR$(99):FORK=1TO253-SR-1
75 DN$=DN$+CHR$(32):NEXTK
77 REM CREA FILE USER
79 REM ***************
81 PRINT"MONTA DISCO DATI"
83 PRINT"PREMI UN TASTO PER PROSEGUIRE"
85 R$="":GETR$:IFR$=""THEN85
87 PRINT"PAZIENZA, ATTENDI, STO PREPARANDO"
89 PRINT"IL DISCO DATI"
91 CLOSE15:OPEN15,8,15,"N0:"+N$+",99"
93 REM RIEMPIE RECORD
95 REM CON CAMPI CHE INIZIANO CON CHR$(99)
97 REM ************************************
99 OPEN2,8,2,N$+",U,W":GOSUB229
101 REM SCRIVE PRIMO RECORD PER DATI DISCO
103 REM OCCUPANDO UN INTERO SETTORE
105 REM ****************************
107 PRINT#2,N$CH$"GGMMAA"CH$NX$CH$"0"CH$" "CH$"
"CH$;
109 GOSUB229
111 PRINT#2,S1$CH$S1$CH$S1$CH$S1$CH$S1$CH$S2$
113 GOSUB229
115 FORK=1TON
117 FORJ=1TONC
119 PRINT#2,Y$(J);CH$;
121 GOSUB229
123 NEXTJ
125 IFSR<254THENPRINT#2,DN$
127 NEXTK
```

```
129 CLOSE2
131 REM CREA FILE INDICE
133 REM INDIRIZZO PRIMO SETTORE DIRECTORY
135 REM ******************************
137 CLOSE15:OPEN15,8,15,"I"
139 OPEN2,8,2,"#":GOSUB229
141 REM LEGGE SETTORE 18,1
143 REM ****************
145 PRINT#15,"U1:2,0,18,1":GOSUB229
147 REM PUNTATORE A TRACCIA E SETTORE
149 REM ***************************
151 PRINT#15,"B-P:2,3":GOSUB229
153 T$="":S$="":GET#2,T$,S$
155 T$=T$+CHR$(0):S$=S$+CHR$(0)
157 CLOSE2
159 T=ASC(T$):S=ASC(S$)
161 PRINT"PRIMO BLOCCO"
163 PRINT"TRACCIA: ";T;"  SETTORE: ";S
165 PRINT"PREMI UN TASTO PER PROSEGUIRE"
167 R$="":GETR$:IFR$=""THEN167
169 REM GENERA INDICE
171 REM LEGGENDO FILE USR COME RANDOM
173 REM ***************************
175 CLOSE15:OPEN15,8,15,"I"
177 OPEN2,8,2,"#":GOSUB229
179 OPEN3,8,3,"INDI1,S,W":GOSUB229
181 REM SCRIVE PRIMO RECORD INDI1
183 REM ***********************
185 PRINT#3,Y$(1)+Y$(2);CH$;T;CH$;S:GOSUB229
187 REM SCRIVE ALTRI N RECORD
189 REM ********************
191 FORK=1TON
193 PRINT#15,"U1:2,0";T;S:GOSUB229
195 PRINT#15,"B-P:2,0":GOSUB229
197 GET#2,T$,S$:GOSUB229
199 T=ASC(T$+CHR$(0)):S=ASC(S$+CHR$(0))
201 PRINTT,S
203 PRINT#3,Y$(1)+Y$(2);CH$;T;CH$;S:GOSUB229
205 NEXTK:CLOSE2:CLOSE3
207 REM FILE STRUTTURA SU DISCO DATI
209 REM ***************************
211 OPEN2,8,2,"STRUTTURA,S,W":GOSUB229
213 PRINT#2,N$:GOSUB229
215 PRINT#2,NC:GOSUB229
217 FORK=1TONC:PRINT#2,D$(K):GOSUB229:NEXTK
219 FORK=1TONC:PRINT#2,L(K):GOSUB229:NEXTK
221 CLOSE2
223 PRINT"IL DISCO DATI E' PRONTO"
225 CLOSE15
227 OPEN15,8,15,"V":CLOSE15:STOP
229 REM ROUTINE ERRORE
231 REM *************
233 INPUT#15,EN,EM$,ET,ES
```

```
235 IFEN=0THENRETURN
237 PRINT"ERRORE DISCO":PRINTEN,EM$,ET,ES
239 STOP
```

COMMENTO A INIZIOFILE

.1/51: definisce costanti e variabili, legge il file STRUTTURA dal disco dei programmi, chiede il numero N dei record e controlla che non superi 100 (linea 31, potrai eventualmente modificare tale numero). Controlla che la somma delle lunghezze dei campi non superi 254 e se va bene prosegue.

.53/75: prepara i campi dummy con primo carattere CHR$(99).

.77/91: chiede di montare il disco dati e lo formatta.

.93/129: apre il file sequenziale di tipo USR, scrive i dati generali del disco sul primo settore e poi tutti i settori necessari per gli N record, chiude il file.

.131/167: per poter creare il file INDI1 va a leggere dalla directory l'indirizzo del primo blocco del file USR generato.

.169/227: crea il file INDI1, con chiavi dummy e gli indirizzi di traccia e settore, che ricava dai primi due byte di concatenamento dei settori del file sequenziale creato. Scrive sul disco dei dati il file STRUTTURA. Alla fine esegue il comando VALIDATE per essere sicuro della corrispondenza della BAM alla situazione creata. Il primo record di INDI1 contiene l'indirizzo del settore del disco usàto per le informazioni generali sull'archivio.

.229/239: routine errore disco.

Questo programma, a parte la sua utilita' per realizzare il nostro archivio, e' molto interessante per imparare a maneggiare bene le registrazioni che si trovano sul floppy.

Se chiedi la lista della directory del floppy, vedrai registrati 3 file, quello di tipo USR con il nome che gli hai assegnato, INDI1 e STRUTTURA.

```
1 REM ARANDOMUSER
3 REM ***********
5 REM DEFINIZIONE COSTANTI E VARIABILI
7 REM ********************************
9 S1$="DATA ULTIMO AGG.        "
11 S2$="FINITO SPAZIO ASSEGNATO"
13 SP$="":FORK=1TO79:SP$=SP$+" ":NEXTK
15 CH$=CHR$(13):LIM$=CHR$(99)+CHR$(32)+CHR$(32)
17 REM LEGGE DATI DA FILE STRUTTURA
19 REM ***************************
21 PRINT"♥QQ"TAB(8)"MONTA DISCO DATI"
23 GOSUB257
```

```
25 OPEN15,8,15,"I"
27 OPEN2,8,2,"STRUTTURA,S,R":GOSUB219
29 INPUT#2,N$,NC:GOSUB219
31 DIMD$(NC):DIMY$(NC):DIML(NC)
33 FORK=1TONC:INPUT#2,D$(K):NEXTK
35 FORK=1TONC:INPUT#2,L(K):NEXTK
37 CLOSE2
45 REM LEGGE DATI DISCO
47 REM ****************
49 OPEN10,8,10,"INDI1,S,R":GOSUB219
51 REM SETTORE DATI GENERALI
53 REM *********************
55 INPUT#10,A$,TD,SD
57 GOSUB187
59 REM DIMENSIONAMENTI PER INDICE
61 REM *************************
63 DIMC$(N),T%(N),S%(N)
65 REM LETTURA INDICE
67 REM **************
69 GOSUB213:GOSUB231
71 REM PRESENTAZIONE MENU' E SCELTE
73 REM ***************************
75 PRINT"♥QQ";TAB(10);"GESTIONE ARCHIVIO":PRINT
77 PRINTTAB(10)"1=AGGIORNAMENTO"
79 PRINTTAB(10)"2=LISTA PRINCIPALE"
81 PRINTTAB(10)"3=CREAZ.IND.SEC."
83 PRINTTAB(10)"4=LISTA SECONDARIA"
85 PRINTTAB(10)"9=FINE"
87 INPUT"COSA SCEGLI ";X
89 IFX<1ORX>4ANDX<>9THENPRINT"□";:GOTO87
91 PRINT:PRINT"PRESENTI ";K;" RECORD"
93 PRINTS1$;G1$;"/";M1$;"/";A1$
95 GOSUB257
97 IFX=9THEN497
99 ONXGOTO295,427,459,479
101 STOP
103 REM INGRESSO DATI
105 REM *************
107 PRINT"♥INGRESSO DATI"
109 FORJ=1TONC:Y$(J)="":NEXTJ
111 PRINT"PER USCIRE $ PER COGNOME";
113 PRINT"▮SE MANCANO DATIPREMI SOLO RETURN"
115 PRINTD$(1);:INPUTY$(1)
117 IFY$(1)="$"THENW=1:RETURN
119 FORJ=2TONC:PRINTD$(J);:INPUTY$(J):NEXTJ
121 REM SISTEMA LUNGHEZZA DATI
123 REM **********************
125 FORJ=1TONC:Y$(J)=LEFT$(Y$(J)+SP$,L(J))
127 NEXTJ:GOSUB371:RETURN
129 REM SCRITTURA INDICE PRINCIPALE
131 REM ***************************
133 PRINT#15,"S0:INDI1":PRINT#15,"I"
135 OPEN10,8,10,"INDI1,S,W":GOSUB219
```

```
137 PRINT#10,LIM$;CH$;TD;CH$;SD;CH$;:GOSUB219
139 FORJ=1TON
141 PRINT#10,C$(J);CH$;T%(J);CH$;S%(J);CH$;
143 GOSUB219:NEXTJ
145 CLOSE10:RETURN
147 REM SCRITTURA NEL BUFFER
149 REM ********************
151 FORJ=1TONC:Y$(J)=LEFT$(Y$(J)+SP$,L(J))
153 PRINT#11,Y$(J);CH$;:GOSUB219
155 NEXTJ:RETURN
157 REM LETTURA SETTORE NEL BUFFER
159 REM **************************
161 PRINT#15,"U1:11,0";T;S:GOSUB219:RETURN
163 REM PUNTATORE NEL BUFFER
165 REM ********************
167 PRINT#15,"B-P:"11;2:GOSUB219:RETURN
169 REM SCRITTURA RECORD
171 REM ****************
173 PRINT#15,"U2:"11;0;T;S:GOSUB219:RETURN
175 REM LETTURA RECORD
177 REM **************
179 PRINT#15,"I":OPEN11,8,11,"#":GOSUB219
181 PRINT#15,"U1:"11;0;T;S:GOSUB219:GOSUB163
183 FORJ=1TONC:INPUT#11,Y$(J):GOSUB219:NEXTJ
185 NEXTJ:CLOSE11:RETURN
187 REM LETT. DATI DISCO
189 REM ****************
191 OPEN11,8,11,"#":GOSUB219
193 PRINT#15,"U1:"11;0;TD;SD:GOSUB219
195 GOSUB163:INPUT#11,N$,R$,N,K:GOSUB219
197 G1$=LEFT$(R$,2):M1$=MID$(R$,3,2)
199 A1$=RIGHT$(R$,2):CLOSE11:RETURN
201 REM LETT. INDICE
203 REM ************
205 CLOSE10:OPEN10,8,10,"INDI1,S,R":GOSUB219
207 INPUT#10,A$,TD,SD
209 REM DA SECONDO RECORD
211 REM *****************
213 FORJ=1TON
215 INPUT#10,C$(J),T%(J),S%(J):GOSUB219
217 NEXTJ:CLOSE10:RETURN
219 REM ROUTINE ERRORE
221 REM **************
223 INPUT#15,EN,EM$,ET,ES
225 IFEN=0THENRETURN
227 PRINT"♥ERRORE DISCO"
229 PRINTEN,EM$,ET,ES:CLOSE15:STOP
231 REM DATA DISCO
233 REM **********
235 PRINT"DATA PER DISCO"
237 INPUT"  GG,MM,AA████████████";G$,M$,A$:RETURN
239 REM SCRITT. IND. SEC.
241 REM *****************
```

```
243 PRINT#15,"S:INDI2":PRINT#15,"I"
245 OPEN10,8,10,"INDI2,S,W"
247 FORJ=1TOK
249 PRINT#10,C$(J);CH$;T%(J);CH$;S%(J);CH$;
251 GOSUB219:NEXTJ:CLOSE10:RETURN
253 REM ATTESA TASTO
255 REM ************
257 R$="":GETR$:IFR$=""THEN257
259 RETURN
261 REM RICERCA POSTO LIBERO
263 REM ********************
265 FORI=1TON
267 IFLEFT$(C$(I),1)<>CHR$(99)THEN271
269 T=T%(I):S=S%(I):JJ=I:I=N
271 NEXTI:RETURN
273 REM AGGIUNTA NUOVI RECORD
275 REM *********************
277 GOSUB103:GOSUB261
279 IFW=1THENK=K-1:CLOSE11:GOTO497
281 GOSUB157:GOSUB163:GOSUB147
283 REM AGGIORNAMENTO INDICE
285 REM ********************
287 C$(JJ)=Y$(1)+Y$(2)
289 GOSUB169:K=K+1
291 IFK>NTHENK=K-1:PRINTS2$:CLOSE11:GOTO497
293 GOTO277
295 REM AGGIORNAMENTO
297 REM **************
299 PRINTTAB(10);"[CLR][DOWN][DOWN]AGGIORNAMENTO ARCHIVIO"
301 PRINT:PRINTTAB(10);"1=CORREZIONE"
303 PRINTTAB(10);"2=AGGIUNTA ELEM."
305 PRINTTAB(10);"3=CANCELL.ELEM."
307 PRINTTAB(10);"9=FINE"
309 INPUT"COSA SCEGLI ";X:IFX=9THEN497
311 IFX<1ORX>3THEN299
313 IFX=2THEN393
315 IFX=3THEN409
317 GOTO351
319 REM RICERCA RECORD
321 REM ***************
323 PRINT"[CLR]";D$(1);:Y$(1)="":INPUTY$(1):W=0
325 IFY$(1)="$"THENW=1:RETURN
327 PRINTD$(2);:Y$(2)="":INPUTY$(2)
329 Y$(1)=LEFT$(Y$(1)+SP$,L(1))
331 Y$(2)=LEFT$(Y$(2)+SP$,L(2))
333 INPUT"QUALE OCCORRENZA ";X
335 IFX<=0THENPRINT"[UP]";:GOTO333
337 FORJ=1TON:IFC$(J)=Y$(1)+Y$(2)THEN345
339 NEXTJ
341 J=J-1:PRINT"NON TROVATO ";Y$(1);"  ";Y$(2)
343 GOSUB257:GOTO319
345 IFX<>1THENX=X-1:GOTO339
347 T=T%(J):S=S%(J)
```

```
349 I=J:J=N:NEXTJ:RETURN
351 REM CORREZIONE
353 REM **********
355 GOSUB319:IFW=1THEN299
357 GOSUB175
359 REM VA A MODIFICA DATI
361 REM ******************
363 GOSUB371
365 PRINT#15,"I":OPEN11,8,11,"#":GOSUB163:GOSUB1
47
367 C$(I)=Y$(1)+Y$(2):T%(I)=T:S%(I)=S
369 GOSUB169:CLOSE11:GOTO351
371 REM CONTROLLO DATI E MODIFICHE
373 REM **************************
375 PRINT"□CONTROLLO DATI E MODIFICHE"
377 FORJ=1TONC:PRINTJ;" ";D$(J);" ";Y$(J):NEXTJ
379 INPUT"□TUTTO BENE S/N ";X$:IFX$="S"THENRETUR
N
381 PRINT"QUALE CAMPO (0 USCITA)";:INPUTX
383 IFX=0THENRETURN
385 IFX>NCORX<1THENPRINT"□□":GOTO379
387 Y$(X)=""
389 INPUTY$(X):Y$(X)=LEFT$(Y$(X)+SP$,L(X))
391 GOTO375
393 REM AGGIUNTA
395 REM ********
397 OPEN11,8,11,"#":GOSUB219
399 PRINT"□AGGIUNTA NUOVI RECORD"
401 PRINT"PRESENTI ";K;" RECORD"
403 PRINT"PUOI AGGIUNGERE ";N-K;" RECORD"
405 GOSUB257
407 W=0:K=K+1:GOTO277
409 REM CANCELLAZIONE
411 REM *************
413 M=0
415 GOSUB319:IFW=1THEN423
417 X=L(1)+L(2):C$(I)=LEFT$(LIM$+SP$+SP$,X)
419 M=M+1
421 GOTO415
423 REM SIST. INDICE
425 GOSUB523:K=K-M:GOTO507
427 REM LISTA FILE
429 REM **********
431 T$="LISTA PER INDICE PRIM."
433 PRINT"ACCENDI STAMPANTE      PREMI UN TASTO"
435 GOSUB257
437 PRINT"□";T$:OPEN4,4
439 PRINT#4:PRINT#4:PRINT#4,T$
441 PRINT#4:PRINT#4
443 FORI=1TOK:T=T%(I):S=S%(I):GOSUB175
445 PRINT#4,Y$(1);"  ";Y$(2)
447 IFNC=2THEN451
449 FORM=3TONC:PRINT#4,D$(M)": "Y$(M):NEXTM
```

```
451 PRINT#4:PRINT#4
455 NEXTI
457 CLOSE4:GOTO517
459 REM CREAZIONE INDICE SECONDARIO
461 REM ***************************
463 PRINT"INDICE SECONDARIO"
465 INPUT"QUALE CAMPO ";X$
467 X=VAL(X$):IFX<2ORX>NCTHENGOTO465
469 FORI=1TOK:T=T%(I):S=S%(I):GOSUB175
470 IFLEN(Y$(X))=0THENY$(X)=CHR$(160)
471 C$(I)=Y$(X):NEXTI:GOSUB551
473 GOSUB239
475 PRINT"FINITO IND. SEC."
477 GOTO517
479 REM LISTA PER INDICE SECONDARIO
481 REM ***************************
483 PRINT"LISTA IND. SEC."
485 OPEN10,8,10,"INDI2,S,R":GOSUB219
487 FORJ=1TOK:INPUT#10,C$(J),T%(J),S%(J)
489 GOSUB219:NEXTJ
491 CLOSE10
493 T$="LISTA IND. SEC. "
495 GOTO433
497 REM CHIUSURA
499 REM ********
501 PRINT"CHIUSURA ARCHIVIO"
503 PRINT"IND. PRINC. DA ORDIN. S/N ":INPUTR$
505 IFR$="S"THENGOSUB523
507 GOSUB129:OPEN11,8,11,"#":GOSUB219
509 PRINT#15,"U1:"11;0;TD;SD:GOSUB219:GOSUB163
511 PRINT#11,N$CH$G$M$A$CH$;N;CH$;K;CH$;
513 GOSUB219
515 PRINT#15,"U2:"11;0;TD;SD:GOSUB219:CLOSE11
517 PRINT"FINITO AGGIORNAMENTO"
519 PRINT"SONO PRESENTI ";K;" RECORD"
521 CLOSE15:STOP
523 REM ORDINAMENTO CRESCENTE
525 REM *********************
527 L=N-1
529 W=0
531 FORJ=1TOL
533 IFC$(J)<=C$(J+1)THEN543
535 R$=C$(J):C$(J)=C$(J+1):C$(J+1)=R$
537 X=T%(J):T%(J)=T%(J+1):T%(J+1)=X
539 X=S%(J):S%(J)=S%(J+1):S%(J+1)=X
541 W=1
543 NEXTJ
545 IFW=0THENRETURN
547 IFL=1THENRETURN
549 L=L-1:GOTO0529
551 REM ORDINAMENTO DECRESCENTE
553 REM **********************
555 L=K-1
```

```
557 W=0
559 FORJ=1TOL
561 IFC$(J)>=C$(J+1)THEN571
563 R$=C$(J):C$(J)=C$(J+1):C$(J+1)=R$
565 X=T%(J):T%(J)=T%(J+1):T%(J+1)=X
567 X=S%(J):S%(J)=S%(J+1):S%(J+1)=X
569 W=1
571 NEXTJ
573 IFW=0THENRETURN
575 IFL=1THENRETURN
577 L=L-1:GOTO0557
```

Non ci sembra necessario riportare il commento di ARANDOMUSER. Esso e' infatti stato ricavato da ARCHIRANDOM, apportando alcune modifiche che elenchiamo:

. il settore che contiene i dati generali del disco non e' piu' quello di indirizzo 1,0; il suo indirizzo si trova nel primo record di INDI1.

. i dati generali del disco non comprendono piu' l'indirizzo dell'ultimo settore scritto, ma e' presente il numero N dei record previsti e il numero K di quelli effettivamente presenti.

. il file INDI1 deve essere trasferito nei 3 vettori ad esso riservati a partire dal secondo record.

. il file INDI1 deve essere sempre ordinato e riscritto tutto, altrimenti si perdono gli indirizzi di traccia e settore.

. i record di INDI1 non usati, chiave che inizia con CHR$(99), nell'ordinamento vanno in fondo, ma restano.

. per cancellare un record si pone CHR$(99) all'inizio della chiave in INDI1.

. se si crea INDI2 si deve lavorare su INDI1 ordinato, INDI2 viene preparato solo per i K record presenti.

. nella fase di aggiunta record si deve cercare in INDI1 la prima chiave che inizia con CHR$(99), leggere il corrispondente settore e riscriverlo senza perdere i primi due caratteri di concatenamento.

. nel programma non e' piu' presente la fase di inizializzazione, svolta dal programma INIZIOFILE.

Se hai riflettuto bene su questo argomento, non ti sara' difficile adattare questi programmi per poter gestire anche record logici di lunghezza sottomultipla di un settore; solo che in questo caso l'indice deve contenere anche un campo che fornisca la posizione del record logico nel settore, da caricare nel puntatore al buffer.

6.7 FILE RELATIVI DI DATI

I file di questo tipo sono trattati dalle istruzioni del DOS per la gestione dei file relativi, elencate nel Paragrafo 6.2.
La differenza tra i file random e i file relativi consiste nei seguenti fatti:

. per accedere ai record di un file relativo si deve fornire al programma il numero d'ordine del record nel file, indirizzo logico, invece dell'indirizzo fisico del blocco.

. i record logici devono essere di lunghezza fissa, che deve essere al massimo di 254 caratteri, comprendendo i caratteri di fine campo e/o record. Pero' la lunghezza puo' essere anche un numero che non risulti sottomultiplo di 254, infatti il sistema gestisce anche record logici registrati a cavallo tra due settori (spanned), come per i file sequenziali.
I primi due caratteri di ogni settore sono usati, come al solito, per concatenare tra loro i settori.

. il sistema, conoscendo la lunghezza di un record e il suo numero d'ordine nel file, ricava con un semplice algoritmo l'indirizzo di traccia e settore necessario.

. il sistema gestisce i file relativi con l'aiuto di un suo indice interno, che genera al momento della creazione del file, chiamato SIDE SECTOR. Esso e' formato al massimo da 6 settori numerati logicamente da 0 a 5, che vengono utilizzati per contenere l'indice di tutti i blocchi fisici utilizzati per il file relativo. In ogni blocco dell'indice SIDE SECTOR possono essere registrati gli indirizzi di 120 settori; per questa ragione il file relativo puo' occupare al massimo 120*6=720 settori, che nel nostro caso sono di piu' di quelli disponibili su un floppy. Inoltre il numero dei record logici gestibili non puo' superare 65535, infatti e' espresso in due byte consecutivi. La struttura di un blocco SIDE SECTOR e' la seguente:

- . byte 0 e 1, concatenamento al settore successivo.
- . byte 2, numero del blocco, da 0 a 5.
- . byte 3, lunghezza del record logico, massimo 254.
- . byte 4 e 5, traccia e settore del SIDE SECTOR 0.
- . byte 6 e 7, traccia e settore del SIDE SECTOR 1.
- . byte 8 e 9, traccia e settore del SIDE SECTOR 2.
- . byte 10 e 11, traccia e settore del SIDE SECTOR 3.
- . byte 12 e 13, traccia e settore del SIDE SECTOR 4.
- . byte 14 e 15, traccia e settore del SIDE SECTOR 5.
- . byte 16 e 17, indirizzo primo settore dati (t,s).
- . byte 18 e 19, indirizzo secondo settore dati (t,s).

.....
.....

- . byte 254 e 255, indirizzo 120-esimo settore dati (t,s).

. viene registrata una entrata nella directory che tiene conto di tutti i blocchi occupati, file e side sector. Inoltre nell'en-

trata e' indicato l'indirizzo del primo settore del side sector.
. il file puo' essere creato scrivendo i record in ordine casuale; in conseguenza possono restare dei record non usati, che il sistema riempie scrivendo 255 (FFH) nella prima posizione e tutti bit 0 nelle altre.
. il sistema usa un buffer in piu' per gestire il file, infatti deve leggere anche il side sector.
. e' consigliabile preestendere il file in fase di creazione; questo rende piu' veloci le operazioni successive. Per preestendere il file basta scrivere l'ultimo record; infatti pensa il sistema a predisporre tutti i precedenti e a creare il numero di blocchi di side sector necessari.

Anche per i file relativi sussiste il problema dell'accesso ai record, gia' esaminato per i file random. L'indice SIDE SECTOR creato dal sistema serve solo per gestire i blocchi fisici. In conseguenza un buon programma di gestione di un archivio deve creare anche un indice sequenziale per chiavi di accesso; in questo caso l'indice puo' essere formato solo da due campi: chiave di accesso, numero d'ordine del record nel file.

Seguono 4 programmi esempio che consentono di capire come si usano i comandi del DOS per i file relativi; essi sono:
. FRELCREA, per preestendere un file,
. FRELSCRIVI, per scrivere record in un file,
. FRELLEGGI, per leggere record da un file,
. FRELCAMPOLEGGI, per mostrare come si puo' accedere a un campo e non a tutto il record.
Questi programmi gestiscono un file relativo di nome PROVA REL, formato da 30 record logici di 21 caratteri ciascuno. Ogni record comprende 2 campi, il primo di 10 caratteri + fine campo (CHR$(13)), e il secondo di 9 caratteri + fine campo.
Dopo aver fatto girare i singoli programmi puoi analizzare il contenuto della directory e dei blocchi occupati per verificare quanto sopra esposto, usando i programmi di utilita' DCOMEFS e TRAC/SET.
Osserva in ogni programma come avviene la preparazione del numero del record e del puntatore al campo.

```
1 REM FRELCREA
3 REM ********
5 REM CREAZIONE FILE RELATIVO
7 REM PREESTENSIONE FILE
9 REM 30 RECORD DI 21 CARATTERI
13 REM COMPRESO RETURN FINALE
15 REM *************************
17 DR$="0":NF$=DR$+":PROVA REL,L,"
19 SA=2:LF=2:RC$="----------":RC$=RC$+RC$
21 BI=1:NR=30:LU=21
23 OPEN15,8,15,"I"
25 REM APERTURA FILE
```

```
27 REM *************
29 OPENLF,8,SA,NF$+CHR$(LU)
31 GOSUB73
33 REM PREESTENSIONE FILE
35 REM PER NR RECORD
37 REM *****************
39 FORX=NRTO1STEP-1
41 HI=INT(X/256):LO=X-HI*256
43 C$="P"+CHR$(SA)
45 REM NUMERO RECORD (BYTE BASSO + BYTE ALTO)
47 REM *************************************
49 C$=C$+CHR$(LO)+CHR$(HI)
51 REM PUNTATORE AL PRIMO BYTE DEL RECORD
53 REM *********************************
55 C$=C$+CHR$(BI)
57 REM POSIZIONA IL PUNTATORE
59 REM *********************
61 PRINT#15,C$:GOSUB73
63 REM SCRIVE IL RECORD DI LINEETTE
65 REM ***************************
67 PRINT#LF,RC$:GOSUB73
69 NEXTX:CLOSELF:GOSUB73:CLOSE15:END
71 REM ROUTINE ERRORE
73 REM **************
75 INPUT#15,EN,EM$,ET,ES
77 IFEN=0OREN=50THENRETURN
79 PRINTEN;EM$;ET;ES
81 STOP:RETURN
```

```
1 REM FRELSCRIVI
3 REM *********
5 REM SCRITTURA RECORD NEL FILE
7 REM RELATIVO, OGNI RECORD 2 CAMPI
9 REM NOME DI 10 CARATTERI
11 REM TELEFONO DI 9 CARATTERI
13 REM LUNGHEZZA RECORD=10+1+9+1=21
15 REM ***************************
17 DR$="0":NF$=DR$+":PROVA REL,L,"
19 LU=21:SA=2:LF=2:B$="          "
21 BI=1:NR=30
23 OPEN15,8,15,"I"
25 REM APERTURA FILE
27 REM *************
29 OPENLF,8,SA,NF$+CHR$(LU)
31 GOSUB71
33 PRINT"NOME=* PER USCIRE":PRINT
35 INPUT"NOME(10)";N$
37 IFN$="*"THEN69
39 N$=LEFT$(N$+B$,10)
41 INPUT"TEL. (9)";T$:T$=LEFT$(T$+B$,9)
```

```
43 INPUT"RECORD ";R
45 IFR=0ORR>NRTHENPRINTCHR$(145);:GOTO43
47 REM PREPARAZIONE PUNTATORE
49 REM *********************
51 HI=INT(R/256):LO=R-HI*256
53 C$="P"+CHR$(SA)
55 C$=C$+CHR$(LO)+CHR$(HI)
57 C$=C$+CHR$(BI)
59 PRINT#15,C$:GOSUB71
61 REM SCRIVE RECORD
63 REM *************
65 PRINT#LF,N$;CHR$(13);T$:GOSUB71
67 GOTO33
69 CLOSELF:GOSUB71:CLOSE15:END
71 REM ROUTINE ERRORE, ACCETTA COD. 50
73 REM ******************************
75 INPUT#15,EN,EM$,ET,ES
77 IFEN=0OREN=50THENRETURN
79 PRINTEN;EM$;ET;ES
81 STOP:RETURN
```

```
1 REM FRELLEGGI
3 REM *********
5 REM LETTURA RECORD
7 REM **************
9 DR$="0":NF$=DR$+":PROVA REL,L,"
11 LU=21:SA=2:LF=2
13 BI=1:NR=30
15 OPEN15,8,15,"I"
17 REM APRE FILE
19 REM *********
21 OPENLF,8,SA,NF$+CHR$(LU)
23 GOSUB61
25 PRINT"RECORD=0 PER USCIRE":PRINT
27 INPUT"RECORD ";R
29 IFR=0THEN59
31 IFR<0ORR>NRTHENPRINTCHR$(145);:GOTO27
33 REM PREPARA PUNTATORE
35 REM *****************
37 HI=INT(R/256):LO=R-HI*256
39 C$="P"+CHR$(SA)
41 C$=C$+CHR$(LO)+CHR$(HI)
43 C$=C$+CHR$(BI)
45 PRINT#15,C$:GOSUB61
47 REM LEGGE RECORD
49 REM ************
51 INPUT#LF,N$,T$:GOSUB61
53 PRINT"NOME: ";N$
55 PRINT"TEL.: ";T$
57 GOTO25
59 CLOSELF:GOSUB61:CLOSE15:END
```

```
61 REM ROUTINE ERRORE, AMMETTE COD. 50
63 REM ******************************
65 INPUT#15,EN,EM$,ET,ES
67 IFEN=0OREN=50THENRETURN
69 PRINTEN;EM$;ET;ES
71 STOP:RETURN
```

```
1 REM FRELCAMPOLEGGI
3 REM **************
5 REM LETTURA SOLO DEL SECONDO CAMPO
7 REM DI UN RECORD R
9 REM ****************************
11 DR$="0":NF$=DR$+":PROVA REL,L,"
13 LU=21:SA=2:LF=2
15 BI=12:NR=30
17 INPUT"RECORD ";R
19 IFR=0ORR>NRTHENPRINTCHR$(145);:GOTO15
21 OPEN15,8,15,"I"
23 REM APREFILE
25 REM ********
27 OPENLF,8,SA,NF$+CHR$(LU)
29 GOSUB55
31 REM PREPARA PUNTATORE
33 REM *****************
35 HI=INT(R/256):LO=R-HI*256
37 C$="P"+CHR$(SA)
39 C$=C$+CHR$(LO)+CHR$(HI)
41 C$=C$+CHR$(BI)
43 PRINT#15,C$:GOSUB55
45 REM LEGGE CAMPO
47 REM ***********
49 INPUT#LF,T$:GOSUB55
51 PRINT"TEL.: ";T$
53 CLOSELF:GOSUB55:CLOSE15:END
55 REM ROUTINE ERRORE, AMMETTE COD. 50
57 REM ******************************
59 INPUT#15,EN,EM$,ET,ES
61 IFEN=0OREN=50THENRETURN
63 PRINTEN;EM$;ET;ES
65 STOP:RETURN
```

Ti facciamo notare che la routine di errore scarta il codice 50, che in questo caso non e' un errore.
Il programma FLCREA preestende il file senza dividere il record in campi. Il puntatore deve avere il valore 1 per iniziare a scrivere il record.
Nel programma FRELSCRIVI il record viene scritto con una sola PRINT# che ha nella lista-dati tutti i campi con i loro separatori.

Il programma FRELLEGGI legge i due campi del record con una sola INPUT# in due variabili.
Nel programma FRELCAMPOLEGGI invece il puntatore viene posto a 12 per leggere solo il secondo campo del record.

Abbiamo modificato il programma ARCHIRANDOM per ottenere ARCHIRELATIVO, che gestisce un file relativo con indice principale e secondario. In questo caso i dati generali dell'archivio, che erano memorizzati nel settore 0 della traccia 1, sono memorizzati in un piccolo file sequenziale di nome DATIGEN, sul disco dati, che viene creato quando si crea ex-novo l'archivio, e, ad ogni elaborazione, viene letto all'inizio e aggiornato alla fine. Per il resto abbiamo lasciato invariata la struttura del record e dei campi. Inoltre, nella fase di creazione dell'archivio, i record sono caricati in sequenza, senza chiedere il numero del record, come e' stato fatto in FRELSCRIVI.

```
1 REM ARCHIRELATIVO
3 REM *************
5 REM DEFINIZIONE COSTANTI E VARIABILI
7 REM *********************************
9 NC=14:REM NUMERO CAMPI RECORD
11 S1$="DATA ULTIMO AGG.      "
13 S2$="FINITO SPAZIO ASSEGNATO"
15 SP$="":FORK=1TO79:SP$=SP$+" ":NEXTK
17 SWW=0:REM PER EVITARE RIDIMENSIONAMENTO
19 CH$=CHR$(13):LIM$=CHR$(99)+CHR$(99)+CHR$(99)
21 DIMD$(NC):DIMY$(NC):DIML(NC)
23 DATA"COGNOME:","NOME   :","INDIR. :"
25 DATA"CITTA' :","PROV.  :","TELEF. :"
27 DATA"L.NASC.:","D.NASC.:","T.STUD.:"
29 DATA"OCC.AT.:","OCC.PR.:","ST.CIV.:"
31 DATA"NOTA 1 :","NOTA 2 :"
33 DATA20,15,30,25,11,10,20,8,20,20,20,1,20,20
35 FORJ=1TONC:READD$(J):NEXTJ
37 FORJ=1TONC:READL(J):NEXTJ
39 REM PRESENTAZIONE MENU' E SCELTE
41 REM *****************************
43 CLOSE15:OPEN15,8,15
45 PRINT"□□□";TAB(10);"GESTIONE ARCHIVIO":PRINT
47 PRINTTAB(10)"1=INIZIO EX-NOVO"
49 PRINTTAB(10)"2=AGGIORNAMENTO"
51 PRINTTAB(10)"3=LISTA PRINCIPALE"
53 PRINTTAB(10)"4=CREAZ.IND.SEC."
55 PRINTTAB(10)"5=LISTA SECONDARIA"
57 PRINTTAB(10)"9=FINE"
59 INPUT"COSA SCEGLI ";X
61 IFX<1ORX>5ANDX<>9THENPRINT"□";:GOTO59
63 PRINT:GOSUB297
65 R$="":PRINTTAB(10)"MONTA DISCO DATI"
```

```
67 GETR$:IFR$=""THEN67
69 IFX=1THEN91
71 PRINT#15,"I"
73 GOSUB259:PRINT"PRESENTI ";K;" RECORD"
75 NN$=N$+",L,"+CHR$(254)
77 PRINTS1$;G1$;"/";M1$;"/";A1$
79 R$="":GETR$:IFR$=""THEN79
81 IFSWW=0THENDIMC$(N),T%(N)
83 GOSUB273
85 IFX=9THEN561
87 ONX-1GOTO361,491,523,543
89 STOP
91 REM INIZIO EX-NOVO ARCHIVIO
93 REM **********************
95 PRINT"POSSO FORMATTARE IL DISCO DATI S/N"
97 R$="":INPUTR$:IFR$=""THEN97
99 IFR$="S"THEN103
101 STOP
103 GOSUB323
105 INPUT"QNOME FILE DA INIZIARE: ";N$
107 INPUT"QUANTI RECORD IN TUTTO: ";N
109 DIMC$(N),T%(N):SWW=1
111 REM PREPARAZIONE CAMPI DUMMY
113 REM ************************
115 P1$="*":FORJ=1TO52:P1$=P1$+" ":NEXTJ
117 P2$=LEFT$(P1$,37)
119 REM PREESTENSIONE FILE N$
121 REM *********************
123 OPEN11,8,11,N$+",L,"+CHR$(254)
125 FORJ=NTO1STEP-1
127 HI=INT(J/256):LO=J-HI*256
129 C$="P"+CHR$(11)+CHR$(LO)+CHR$(HI)+CHR$(1)
131 PRINT#15,C$:GOSUB285
133 PRINT#11,P1$CH$P1$CH$P1$CH$P1$CH$P2$CH$;
135 GOSUB285
137 NEXTJ:CLOSE11:GOSUB285
139 REM PREPARAZIONE INDICE
141 REM ******************
143 FORJ=1TON:T%(J)=J
145 C$(J)=LIM$:NEXTJ
147 GOSUB177:REM SCRIVE INDI1
149 K=0:GOSUB221:GOTO43
151 REM INGRESSO DATI
153 REM *************
155 PRINT"LINGRESSO DATI"
157 FORJ=1TONC:Y$(J)="":NEXTJ
159 PRINT"PER USCIRE $ PER COGNOME";
161 PRINT"USE MANCANO DATIPREMI SOLO RETURN"
163 PRINTD$(1);:INPUTY$(1)
165 IFY$(1)="$"THENW=1:RETURN
167 FORJ=2TONC:PRINTD$(J);:INPUTY$(J):NEXTJ
169 REM SISTEMA LUNGHEZZA DATI
171 REM *********************
```

```
173 FORJ=1TONC:Y$(J)=LEFT$(Y$(J)+SP$,L(J))
175 NEXTJ:GOSUB439:RETURN
177 REM SCRITTURA INDICE PRINCIPALE
179 REM **************************
181 PRINT#15,"S0:INDI1":PRINT#15,"I"
183 OPEN10,8,10,"INDI1,S,W":GOSUB285
185 FORJ=1TON
187 PRINT#10,C$(J);CH$;T%(J);CH$;
189 GOSUB285:NEXTJ
191 CLOSE10:RETURN
193 REM SCRITTURA RECORD PUNTATO
195 REM COSTRUZIONE STRINGA DI STAMPA
197 REM ****************************
199 C$="":FORJ=1TONC
201 Y$(J)=LEFT$(Y$(J)+SP$,L(J))
203 C$=C$+Y$(J)+CH$:NEXTJ
205 PRINT#11,C$;:GOSUB285
207 RETURN
209 REM DEFINISCE POSIZIONE RECORD K
211 REM ***************************
213 HI=INT(T/256)
215 LO=T-HI*256
217 PRINT#15,"P"+CHR$(11)+CHR$(LO)+CHR$(HI)+CHR$
(1)
219 GOSUB285:RETURN
221 REM SCRITTURA DATIGEN DISCO
223 REM **********************
225 PRINT#15,"S0:DATIGEN"
227 PRINT#15,"I"
229 OPEN10,8,10,"DATIGEN,S,W":GOSUB285
231 PRINT#10,N$CH$G$M$A$CH$;N;CH$;K;CH$;
233 CLOSE10:RETURN
235 REM RICERCA POSTO NELL'INDICE
237 REM ***********************
239 JJ=0:FORJ=1TON
241 IFLEFT$(C$(J),1)=CHR$(99)THEN 245
243 NEXTJ:RETURN
245 JJ=J:J=N:GOTO243
247 REM LETTURA RECORD
249 REM *************
251 PRINT#15,"I":OPEN11,8,11,NN$:GOSUB285
253 GOSUB209
255 FORJ=1TONC:INPUT#11,Y$(J):GOSUB285
257 NEXTJ:CLOSE11:RETURN
259 REM LETT.DATI DISCO
261 REM **************
263 OPEN10,8,10,"DATIGEN,S,R":GOSUB285
265 INPUT#10,N$,R$,N,K:GOSUB285
267 CLOSE10:GOSUB285
269 G1$=LEFT$(R$,2):M1$=MID$(R$,3,2)
271 A1$=RIGHT$(R$,2):RETURN
273 REM LETT. INDICE
275 REM ***********
```

```
277 OPEN10,8,10,"INDI1,S,R":GOSUB285
279 FORJ=1TON
281 INPUT#10,C$(J),T%(J):GOSUB285
283 NEXTJ:CLOSE10:RETURN
285 REM ROUTINE ERRORE
287 REM *************
289 INPUT#15,EN,EM$,ET,ES
291 IFEN=0OREN=50THENRETURN
293 PRINT"█ERRORE DISCO"
295 PRINTEN,EM$,ET,ES:CLOSE15:STOP
297 REM DATA DISCO
299 REM **********
301 PRINT"DATA PER DISCO"
303 INPUT"  GG,MM,AA███████████";G$,M$,A$:RETURN
305 REM SCRITT. IND. SEC.
307 REM *****************
309 PRINT#15,"S:INDI2":PRINT#15,"I"
311 OPEN10,8,10,"INDI2,S,W"
313 FORJ=1TOK
315 PRINT#10,C$(J);CH$;T%(J);CH$;
317 GOSUB285:NEXTJ:CLOSE10:RETURN
319 R$="":GETR$:IFR$=""THEN319
321 RETURN
323 REM INIZIALIZZAZIONE DISCO
325 REM **********************
327 PRINT"NOME DISCO";:INPUTN$
329 PRINT#15,"N0:"+N$+",99"
331 CLOSE15:OPEN15,8,15:PRINT#15,"I"
333 RETURN
335 REM NUOVI DATI
337 REM **********
339 OPEN11,8,11,NN$:GOSUB285
341 GOSUB151
343 IFW=1THENK=K-1:CLOSE11:GOTO561
345 GOSUB235:IFJJ=0THENSTOP:REM STOP IMPOSS.
347 T=T%(JJ):GOSUB209:GOSUB193
349 REM AGGIORNAMENTO INDICE
351 REM ********************
353 C$(JJ)=Y$(1)+Y$(2)
355 K=K+1
357 IFK>NTHENK=K-1:PRINTS2$:CLOSE11:GOTO561
359 GOTO341
361 REM AGGIORNAMENTO
363 REM **************
365 PRINTTAB(10);"███AGGIORNAMENTO ARCHIVIO"
367 PRINT:PRINTTAB(10);"1=CORREZIONE"
369 PRINTTAB(10);"2=AGGIUNTA ELEM."
371 PRINTTAB(10);"3=CANCELL.ELEM."
373 PRINTTAB(10);"9=FINE"
375 INPUT"COSA SCEGLI ";X:IFX=9THEN561
377 IFX<1ORX>3THEN365
379 IFX=2THEN461
381 IFX=3THEN475
```

```
383 GOTO417
385 REM RICERCA RECORD
387 REM **************
389 PRINT"□";D$(1);:Y$(1)="":INPUTY$(1):W=0
391 IFY$(1)="$"THENW=1:RETURN
393 PRINTD$(2);:Y$(2)="":INPUTY$(2)
395 Y$(1)=LEFT$(Y$(1)+SP$,L(1))
397 Y$(2)=LEFT$(Y$(2)+SP$,L(2))
399 INPUT"QUALE OCCORRENZA ";X
401 IFX<=0THENPRINT"□";:GOTO399
403 FORJ=1TOK:IFC$(J)=Y$(1)+Y$(2)THEN411
405 NEXTJ
407 J=J-1:PRINT"NON TROVATO ";Y$(1);"  ";Y$(2)
409 GOSUB319:GOTO0385
411 IFX<>1THENX=X-1:GOTO0405
413 T=T%(J)
415 I=J:J=K:NEXTJ:RETURN
417 REM CORREZIONE
419 REM **********
421 GOSUB385:IFW=1THEN365
423 GOSUB247
425 REM VA A MODIFICA DATI
427 REM ******************
429 GOSUB439
431 PRINT#15,"I":OPEN11,8,11,NN$
433 GOSUB209:GOSUB193
435 C$(I)=Y$(1)+Y$(2)
437 CLOSE11:GOTO0417
439 REM CONTROLLO DATI E MODIFICHE
441 REM **************************
443 PRINT"□CONTROLLO DATI E MODIFICHE"
445 FORJ=1TONC:PRINTJ;" ";D$(J);" ";Y$(J):NEXTJ
447 INPUT"□TUTTO BENE S/N ";X$:IFX$="S"THENRETUR
N
449 PRINT"QUALE CAMPO (0 USCITA)";:INPUTX
451 IFX=0THENRETURN
453 IFX>NCORX<1THENPRINT"□□":GOTO0447
455 Y$(X)=""
457 INPUTY$(X):Y$(X)=LEFT$(Y$(X)+SP$,L(X))
459 GOTO0443
461 REM AGGIUNTA
463 REM**********
465 PRINT"□AGGIUNTA NUOVI RECORD"
467 PRINT"PRESENTI ";K;" RECORD"
469 PRINT"PUOI AGGIUNGERE ";N-K;" RECORD"
471 GOSUB319
473 W=0:K=K+1:GOTO0335
475 REM CANCELLAZIONE
477 REM *************
479 M=0
481 GOSUB385:IFW=1THEN487
483 X=L(1)+L(2):C$(I)=LEFT$(LIM$+SP$+SP$,X)
485 M=M+1:GOTO0481
```

```
487 REM SIST. INDICE
489 GOSUB581:K=K-M:GOTO573
491 REM LISTA FILE
493 REM **********
495 T$="LISTA PER INDICE PRIM."
497 PRINT"ACCENDI STAMPANTE     PREMI UN TASTO"
499 GETR$:IFR$=""THEN499
501 PRINT"♥";T$:OPEN4,4
503 PRINT#4:PRINT#4:PRINT#4,T$
505 FORJ=1TO10:PRINT#4:NEXTJ
507 L=2:FORI=1TOK:T=T%(I):GOSUB247
509 PRINT#4,Y$(1);"  ";Y$(2)
511 PRINT#4,Y$(3);"  ";Y$(4);"  ";Y$(5)
513 FORM=6TONC:PRINT#4,D$(M);Y$(M):NEXTM
515 PRINT#4:PRINT#4
517 IFL=5THENPRINT#4:L=0
519 L=L+1:NEXTI
521 CLOSE4:GOTO575
523 REM CREAZIONE INDICE SECONDARIO
525 REM ***************************
527 PRINT"♥INDICE SECONDARIO"
529 INPUT"QQUALE CAMPO ";X$
531 X=VAL(X$):IFX<2ORX>NCTHENGOTO529
533 FORI=1TOK:T=T%(I):GOSUB247
534 IFLEN(Y$(X))=0THENY$(X)=CHR$(160)
535 C$(I)=Y$(X):NEXTI
537 GOSUB305:GOSUB273
539 PRINT"FINITO IND. SEC."
541 GOTO575
543 REM LISTA PER INDICE SECONDARIO
545 REM ***************************
547 PRINT"♥LISTA IND. SEC."
549 OPEN10,8,10,"INDI2,S,R":GOSUB285
551 FORJ=1TOK:INPUT#10,C$(J),T%(J)
553 GOSUB285:NEXTJ
555 CLOSE10:GOSUB607:GOSUB305
557 T$="LISTA IND. SEC. "
559 GOTO497
561 REM CHIUSURA
563 REM *********
565 PRINT"CHIUSURA ARCHIVIO"
567 PRINT"♥Q ORDINA INDICE INDI1"
569 PRINT" ATTENDI CON PAZIENZA"
571 GOSUB581
573 GOSUB177:GOSUB221
575 PRINT"FINITO AGGIORNAMENTO"
577 PRINT"SONO PRESENTI ";K;" RECORD"
579 CLOSE15:STOP
581 REM ORDINAMENTO CRESCENTE
583 REM *********************
585 L=N-1
587 W=0
589 FORJ=1TOL
```

```
591 IFC$(J)<=C$(J+1)THEN599
593 R$=C$(J):C$(J)=C$(J+1):C$(J+1)=R$
595 X=T%(J):T%(J)=T%(J+1):T%(J+1)=X
597 W=1
599 NEXTJ
601 IFW=0THENRETURN
603 IFL=1THENRETURN
605 L=L-1:GOTO587
607 REM ORDINAMENTO DECRESCENTE
609 REM **********************
611 L=K-1
613 W=0
615 FORJ=1TOL
617 IFC$(J)>=C$(J+1)THEN625
619 R$=C$(J):C$(J)=C$(J+1):C$(J+1)=R$
621 X=T%(J):T%(J)=T%(J+1):T%(J+1)=X
623 W=1
625 NEXTJ
627 IFW=0THENRETURN
629 IFL=1THENRETURN
631 L=L-1:GOTO613
```

Non riportiamo per esteso il commento al programma, ma segnaliamo le parti che rìguardano il trattamento del file relativo.

. 111/137: preestensione file nella fase 1 del menu'.

. 193/207: viene preparata la stringa di scrittura di un record logico con tutti i campi.

. 209/219: definizione posizione record e puntatore.

. 235/245: ricerca posto nell'indice.

Anche per questo archivio devi fare delle considerazioni sull'occupazione di memoria e la quantita' di record trattabili, come per il file random.

A questo punto pensiamo che lo studio dei diversi programmi di archivio presentati ti potra' garantire una buona conoscenza dei diversi tipi di file gestibili e potrai, o modificare i programmi per adattarli alle tue esigenze, o prepararne altri, magari servendoti di alcune delle routine da noi utilizzate.
In questo capitolo non ci siamo preoccupati di presentare quadri video particolarmente belli, potrai migliorare tu le situazioni, usando anche i colori.
Inoltre, se vuoi, potrai controllare l'ingresso dati, con apposite routine, per evitare segnalazioni di errore che deturpano i quadri video.

6.8 PROGRAMMI DI UTILITA'

Si definiscono tali i programmi che sono di aiuto durante il lavoro di programmazione. Nella scatola dell'unita' 1541 si trova anche il dischetto TEST/DEMO, che contiene alcuni programmi di utilita'. Riportiamo un breve commento ad alcuni di essi.

C-64 WEDGE carica il DOS 5.1; dopo averlo fatto girare sono attivi i seguenti comandi:

. /nome-programma, per caricare un programma dal dischetto.

. > oppure @, per visualizzare lo stato delle 4 variabili di errore disco.

. >$ oppure @$, per visualizzare la directory sul video, senza cancellare il programma presente in memoria.

Il programma resta attivo fino a quando togli corrente.

COPY/ALL ti permette di eseguire la copia dei dischetti collegando 2 unita' 1541. Devi cambiare il "dn" di una di esse da 8 a 9, con il programma DISK ADDR CHANGE. Il cambio di unita' resta valido fino a quando togli corrente. Prima di eseguire una copia conviene proteggere il disco sorgente chiudendo la finestrella laterale.
In commercio esistono programmi che consentono di eseguire la copia dei floppy anche disponendo di una sola unita'; viene richiesto con un messaggio di scambiare i floppy quando e' necessario.

DIR serve per: leggere e listare la directory, eseguire comandi del DOS, visualizzare le 4 variabili di errore. Il programma presenta un certo interesse per le tecniche di programmazione usate. La directory viene aperta come file "$0" sul canale 0, con lfn=1 e poi letta con GET#1, carattere per carattere.

VIEW BAM visualizza sul video la BAM in due quadri video, come una matrice, riportando sull'asse orizzontale le tracce e sull'asse verticale i settori. Il programma contiene un'imperfezione, infatti nelle tracce da 18 a 24 sono mostrati senza l'annullamento N/L, ma come occupati, i settori 19 che non esistono.

CHECK DISK controlla se il dischetto ha settori rovinati, ma risulta molto lento, dopo aver segnalato eventuali settori rovinati lascia il dischetto tutto occupato. Andrebbe eseguito il comando COLLECT o VALIDATE e bisognerebbe escludere i settori rovinati marcandoli come occupati sia nella BAM che nella directory.

DISPLAY T&S serve per visualizzare o su video o su stampante il

contenuto di uno o piu' settori del floppy. Vengono mostrati a sinistra i contenuti esadecimali e a destra i caratteri di stampa. In certi casi va in crisi; puoi premere RESET mentre tieni premuto RUN/STOP, poi uscire dal MONITOR con X e ripartire.

Riportiamo ora alcuni programmi di utilita' preparati da noi.

DCOMEFS stampa la BAM e la directory. Nella prima parte vengono evidenziati i 35 gruppi di 4 byte dedicati alla BAM, vedi descrizione del Paragrafo 6.1. Dopo sono stampate le informazioni relative ad ogni entrata della directory.
Il programma manda l'uscita alla stampante, puoi dirigerla al video modificando la linea 30 in OPEN4,6.

```
5 REM DCOMEFS
10 T$(0)="DEL":T$(1)="SEQ":T$(2)="PRG"
15 T$(3)="USR":T$(4)="REL"
20 CLOSE15:OPEN15,8,15,"I"
25 OPEN2,8,2,"$":REM APERTURA FILE DIRECTORY
30 OPEN4,4:REM APERTURA STAMPANTE
35 PRINT#4,"STAMPA DIRECTORY COME FILE SEQUENZIALE"
40 PRINT#4
45 REM LETTURA PRIMI 2 BYTE
50 I=2:GOSUB230
55 REM LETTURA BAM
60 FORK=1 TO 35:PRINT#4,K;"   ";
65 I=4:GOSUB230
70 NEXTK:PRINT#4
75 REM LETTURA NOME DISCO
80 I=18:GOSUB205
85 PRINT#4,N$;"  ";
90 REM LETTURA ID
95 I=2:GOSUB205:PRINT#4,N$
100 REM LETTURA ALTRI 7 BYTE
105 I=7:GOSUB230
110 PRINT#4
115 REM LETTURA ALTRI 85 BYTE
120 FORK=1TO85:GET#2,A$:NEXTK
125 PRINT#4
130 FORJ=1TO8
135 GET#2,T$,A$,B$
140 IFSTTHENCLOSE2:CLOSE4:CLOSE15:STOP
145 IFT$=""THENT$=CHR$(128)
150 I=16
155 GOSUB205
160 PRINT#4,T$(ASC(T$)-128);"  ";
165 PRINT#4,ASC(A$+CHR$(0));ASC(B$+CHR$(0));
170 PRINT#4,N$
175 I=3:GOSUB230
180 FORK=1TO4:GET#2,A$:NEXTK
185 I=2:GOSUB230
```

```
190 I=2:GOSUB230
195 IFJ<8THENGET#2,A$,A$
200 NEXTJ:GOTO130
205 REM COSTRUZIONE STRINGA
210 N$="":FORL=1TOI
215 GET#2,L$
220 IFL$<>CHR$(96)THENIFL$<>CHR$(160)THENN$=N$+L$
225 NEXTL:RETURN
230 REM LETTURA E STAMPA GRUPPO BYTE
235 FORL=1TOI
240 GET#2,A$:PRINT#4,ASC(A$+CHR$(0));
245 NEXTL:PRINT#4
250 RETURN

READY.
```

CONTRDISCO controlla se sono registrati bene sul disco i file di tipo SEQ e PRG, ma non controlla i file di altro tipo. Ogni volta che controlla un blocco fa apparire un pallino sul video. Se un file non e' in ordine ti chiede se vuoi cancellarlo. Se rispondi S, lo cancella e esegue il comando VALIDATE, se rispondi N si ferma.

```
5 REM CONTRDISCO
10 REM --COSTANTI E VARIABILI--
15 M1$="****************":FI=0
20 M2$="{Q}FILE NON CORRETTO: {Q}"
25 M3$="{Q}TUTTO IN ORDINE   ...{Q} SPERO !"
30 REM --APRE CANALE 15 CON LFN=1--
35 OPEN1,8,15,"I"
40 REM --APRE CANALE 2 PER DIRECTORY--
45 OPEN2,8,2,"#"
50 REM --APRE CANALE 3 PER FILE--
55 OPEN3,8,3,"#"
60 REM --FUNZ. PER PUNTARE ENTRATE DIRECTORY--
65 DEFFNF(X)=2+32*X
70 REM --PRIMO BLOCCO DIRECTORY--
75 TD=18:SD=1:GOSUB350
80 REM --ELABORA UNA ENTRATA--
85 FI$="":CB=0
90 REM --PUNTATORE A INIZIO ENTRATA--
95 PRINT#1,"B-P:2,"FNF(FI)
100 REM --LEGGE TIPO FILE CHE VA IN TY--
105 GET#2,A$:GOSUB385:TY=ASC(A$)
110 REM --LEGGE IND. INIZIO FILE, TR E SC--
115 GET#2,A$,B$:GOSUB380:TR=ASC(A$):SC=ASC(B$)
120 REM --LEGGE NOME FILE--
125 FORI=1TO16:GET#2,A$:FI$=FI$+A$
130 IFA$=CHR$(160)THENFI$=LEFT$(FI$,I-1):I=16
135 NEXT
```

```
140 REM --PUNTATORE AI 2 BYTE CONT. BLOCCHI--
145 PRINT#1,"B-P:2,"FNF(FI)+28:GET#2,A$,B$
150 REM --NB=VCONTATORE BLOCCHI--
155 GOSUB380:NB=ASC(A$)+ASC(B$)*256
160 REM --CONTROLLO TIPO FILE 1 O 2--
165 IFTY<>0THENTY=TY-128
170 IFTY<>2ANDTY<>1THEN310
175 IFTY<>2ANDTY<>1THEN85
180 REM --CONTROLLO BLOCCHI FILE--
185 PRINT"[QQQ]"M1$
190 PRINT"CONTROLLO FILE":PRINTM1$"[QQQ]"
195 PRINTTAB(10)FI$"[QQ]"
200 REM --LEGGE BLOCCO FILE--
205 PRINT#1,"B-R:3,0,"TR,SC
210 REM --LEGGE BYTE CONCATENAMENTO--
215 PRINT#1,"B-P:3,0"
220 CB=CB+1:GET#3,A$,B$:GOSUB380
225 REM --STAMPA UN PALLINO PER OGNI BLOCCO--
230 TR=ASC(A$):SC=ASC(B$):PRINT"●";
235 IFTR<>0THEN205
240 REM --CONTROLLO LUNGHEZZA FILE--
245 IFCB=NBTHEN325
250 REM --CICLO DI ATTESA--
255 IFCB=NBTHENFORI=1TO250:NEXTI:GOTO85
260 PRINT:PRINTM2$FI$
265 REM --SE CONTEGGIO ERRATO--
270 PRINT"[Q]DEVO AZZERARE  ? ([R]S[■]/[R]N[■])"
275 GETA$:IFA$=""THEN275
280 IFA$="N"THEN400
285 IFA$<>"S"THEN275
290 PRINT"[Q]AZZERAMENTO E VALIDATE"
295 PRINT#1,"S:"FI$:PRINT#1,"V"
300 CLOSE3:CLOSE2:CLOSE1:RUN
305 REM --PASSA A PROSSIMA ENTRATA--
310 FI=FI+1:IFFI=8THENGOSUB350:FI=0
315 GOTO175
320 REM --TUTTO BENE--
325 PRINT:PRINTTAB(9)"[QQQ][R]OK":FI=FI+1
330 IFFI=8THENGOSUB350:FI=0
335 GOTO255
340 REM --R[Q]INE LETTURA DIRECTORY--
345 REM --TD=0 A FINE DIRECTORY--
350 IFTD=0THENPRINTM3$:GOTO400
355 REM --LETTURA BLOCCO--
360 A$="":B$="":PRINT#1,"U1:2,0";TD;SD
365 GET#2,A$,B$:GOSUB380
370 TD=ASC(A$):SD=ASC(B$):RETURN
375 REM --SISTEMAZIONE BYTE LETTI--
380 IFB$=""THENB$=CHR$(0)
385 IFA$=""THENA$=CHR$(0)
390 RETURN
395 REM --CHIUSURA FILE--
400 CLOSE3:CLOSE2:CLOSE1:STOP
```

TRAC/SET e' una modifica di DISPLAY T&S e consente di visualizzare gruppi di settori concatenati e settori isolati.

```
5 REM TRAC/SET
10 RC$="     VIDEO          O   PRINTER"
15 RD$="     VIDEO "
20 RE$="     PRINTER"
25 PRINT"   ______________________________"
30 PRINT"  CONTENUTO BLOCCHI   "
35 PRINT"______________________________"
40 REM COSTANTI
45 SP$=" ":NL$=CHR$(0)
50 HX$="0123456789ABCDEF"
55 FS$="":FORI=64 TO 95
60 FS$=FS$+" "+CHR$(I)+" ":NEXT I
65 SS$="  ":FOR I=192 TO 223
70 SS$=SS$+" "+CHR$(I)+" ":NEXT I
75 DIM A$(15),NB(2)
80 D$="0"
85 PRINTRC$
90 GETJJ$:IF JJ$="" THEN90
95 IF JJ$="V"THENPRINTRD$
100 IF JJ$="P"THENPRINTRE$:OPEN4,4
110 GOSUB450
115 REM CARICA BUFFER
120 INPUT"   TRACCIA, SETTORE";T,S
125 IF T=0 OR T>35 THEN355
130 IF JJ$="V" THENGOSUB360:GOTO135
131 PRINT#4:PRINT#4,"TRACCIA"T" SETTORE"S:PRINT#
4
135 PRINT#15,"U1:2,"D$;T;S:GOSUB335
140 PRINT#15,"B-P:2,0"
145 REM LEGGE BYTE 0
150 GET#2,A$(0):IFA$(0)=""THENA$(0)=NL$
155 PRINT#15,"B-P:2,1"
160 IF JJ$="V"THEN170
165 IF JJ$="P"THEN230
170 REM VIDEO
175 K=1:NB(1)=ASC(A$(0))
180 FORJ=0TO63:IFJ=32THENGOSUB375:GOTO365
185 FOR I=K TO 3
190 GET#2,A$(I):IF A$(I)="" THEN A$(I)=NL$
195 IF K=1 AND I<2 THEN NB(2)=ASC(A$(I))
200 NEXT I:K=0
205 A$="":B$=":":N=J*4:GOSUB 405:A$=A$+":"
210 FOR I=0 TO 3:N=ASC(A$(I)):GOSUB 405
215 C$=A$(I):GOSUB 425:B$=B$+C$
220 NEXT I:IF JJ$="V" THEN PRINTA$B$
225 NEXT J:GOTO295
230 REM PRINTER
235 K=1:NB(1)=ASC(A$(0))
240 FOR J=0 TO 15
```

```
245 FOR I=K TO 15
250 GET#2,A$(I):IF A$(I)="" THEN A$(I)=NL$
255 IF K=1 AND I<2 THEN NB(2)=ASC(A$(I))
260 NEXT I:K=0
265 A$="":B$=":":N=J*16:GOSUB 405:A$=A$+":"
270 FOR I=0 TO 15:N=ASC(A$(I)):GOSUB 405
275 C$=A$(I):GOSUB 425:B$=B$+C$
280 NEXT I
285 IF JJ$="P" THEN PRINT#4,A$B$
290 NEXT J:GOTO295
295 REM SUCCESSIVO BLOCCO
300 PRINT"TRACCIA/SETT.SEGUENTE"NB(1)NB(2) ""
305 PRINT"DESIDERI PROSEGUIRE         S/N   "
310 GET Z$:IF Z$="" THEN310
315 IF Z$<>"S" THEN325
320 T=NB(1):S=NB(2):GOSUB445:GOSUB450:GOTO125
325 IF Z$="N" THEN GOSUB445:GOSUB450:GOTO120
330 GOTO 310
335 REM ROUTINE ERRORE
340 INPUT#15,EN,EM$,ET,ES:IF EN=0 THEN RETURN
345 PRINT"ERRORE DISCO"EN,EM$,ET,ES
350 END
355 PRINT#15,"I"D$:GOSUB445:CLOSE4:PRINT"END":EN
D
360 PRINT"TRACCIA"T" SETTORE"S"":RETURN
365 IF Z$="N"THEN J=80:GOTO 225
370 GOTO185
375 REM MESS. CONTINUAZIONE
380 PRINT"CONTINUO(S/N)"
385 GETZ$:IF Z$="" THEN 385
390 IF Z$="N" THEN RETURN
395 IF Z$<>"S" THEN 385
400 PRINT"TRACCIA" T " SETTORE"S:RETURN
405 REM CONV.HEX
410 A1=INT(N/16):A$=A$+MID$(HX$,A1+1,1)
415 A2=INT(N-16*A1):A$=A$+MID$(HX$,A2+1,1)
420 A$=A$+SP$:RETURN
425 REMCONV.ASCII
430 IF ASC(C$)<32 THEN C$=" ":RETURN
435 IF ASC(C$)<128 OR ASC(C$)>159 THEN RETURN
440 C$=MID$(SS$,3*(ASC(C$)-127),3):RETURN
445 CLOSE2:CLOSE15:RETURN
450 OPEN15,8,15,"I"+D$:GOSUB335
455 OPEN2,8,2,"#":GOSUB335:RETURN
```

SIST/DISCO controlla il dischetto per trovare se ci sono settori rovinati; esso e' una modifica di CHECK DISK, ma risulta molto piu' veloce, infatti abbiamo ottimizzato l'allocazione dei blocchi. Alla fine il dischetto risulta con allocati nella BAM i settori rovinati, ma non nella directory, quindi puo' essere usato, ma non puoi eseguire i comandi VALIDATE o COLLECT.

```
5 REM SIST/DISCO
10 REM TABELLA TRACCE E SETTORI GUASTI
15 DIMT(100):DIMS(100)
20 DIMG1(21),G2(19),G3(18),G4(17)
25 DATA0,10,20,8,18,6,16,4,14,2,12,9,19
30 DATA7,17,5,15,3,13,1,11
35 DATA0,11,1,12,2,13,3,14,4,15,5,16,6,17
40 DATA7,18,8,9,10
45 DATA0,10,1,11,2,12,3,13,4,14,5,15
50 DATA6,16,7,17,8,9
55 DATA0,10,1,11,2,12,3,13,4,14,5
60 DATA15,6,16,7,8,9
65 FORJ=1TO21:READG1(J):NEXTJ
70 FORJ=1TO19:READG2(J):NEXTJ
75 FORJ=1TO18:READG3(J):NEXTJ
80 FORJ=1TO17:READG4(J):NEXTJ
85 RC$=" BLOCCHI GUASTI SONO STATI ALLOCATI"
90 RD$="QQBLOCCHI GUASTI":RE$="TRACCIA"
95 RF$="SETTORE"
100 PRINT"♥QQQ——————————————————————"
105 PRINT"      SIST/DISCO      "
110 PRINT"—————————————————————————"
115 OPEN15,8,15
120 PRINT#15,"V0"
125 N%=RND(TI)*255
130 A$="":FORI=1TO255
135 A$=A$+CHR$(255AND(I+N%)):NEXT
140 OPEN2,8,2,"#":GOSUB325
145 PRINT:PRINT#2,A$;
150 J=1
155 FORT=1TO17:FORL=1TO21
160 S=G1(L):GOSUB200:NEXTL:NEXTT
165 FORT=18TO24:FORL=1TO19
170 S=G2(L):GOSUB200:NEXTL:NEXTT
175 FORT=25TO30:FORL=1TO18
180 S=G3(L):GOSUB200:NEXTL:NEXTT
185 FORT=31TO35:FORL=1TO17
190 S=G4(L):GOSUB200:NEXTL:NEXTT
195 GOTO260
200 PRINT#15,"B-A:0"T;S
205 INPUT#15,EN,EM$,ET,ES
210 IFEN=0THEN225
215 IFEN=65THENRETURN
220 STOP
225 PRINT#15,"U2:2,0"T;S
230 NB=NB+1
235 INPUT#15,EN,EM$,ES,ET
240 IF EN=0THENRETURN
245 T(J)=T:S(J)=S:J=J+1
250 PRINT"QQ█BLOCCHI GUASTI:";T;S
255 RETURN
260 PRINT#15,"V0"
265 GOSUB325
```

```
270 CLOSE2
275 IFJ=1THEN350
280 OPEN2,8,2,"#"
285 PRINTRD$,RE$,RF$
295 FORI=1TOJ-1
300 PRINT#15,"B-A:0"T(I);S(I)
305 PRINT,,T(I),S(I)
310 NEXT
315 PRINT"Q";J-1;RC$
320 CLOSE2:CLOSE15:END
325 INPUT#15,EN,EM$,ET,ES
330 IF EN=0 THEN RETURN
335 PRINT"QQ▮ERRORE#"EN,EM$;ET;ES"▮"
340 PRINT#15,"I0"
345 RETURN
350 PRINT"QQQ▮▮TUTTO BENE!    ":CLOSE15:END
```

NUMBUFFER serve per trovare quanti buffer dell'unita' 1541 possono essere aperti contemporaneamente in modo diretto e quali sono i loro numeri.

```
5 REM NUMBUFFER
10 OPEN15,8,15,"I"
15 INPUT"QUANTI BUFFER";N
20 A$="":B$="":C$="":D$="":E$=""
25 ONNGOTO50,45,40,35,30
30 OPEN2,8,2,"#":GOSUB115:GET#2,E$:GOSUB115
35 OPEN3,8,3,"#":GOSUB115:GET#3,D$:GOSUB115
40 OPEN4,8,4,"#":GOSUB115:GET#4,C$:GOSUB115
45 OPEN5,8,5,"#":GOSUB115:GET#5,B$:GOSUB115
50 OPEN6,8,6,"#":GOSUB115:GET#6,A$:GOSUB115
55 ONNGOTO100,90,80,70,60
60 IFE$=""THENE$=CHR$(0)
65 PRINTASC(E$):CLOSE2
70 IFD$=""THEND$=CHR$(0)
75 PRINTASC(D$):CLOSE3
80 IFC$=""THENC$=CHR$(0)
85 PRINTASC(C$):CLOSE4
90 IFB$=""THENB$=CHR$(0)
95 PRINTASC(B$):CLOSE5
100 IFA$=""THENA$=CHR$(0)
105 PRINTASC(A$):CLOSE6
110 CLOSE15:STOP
115 INPUT#15,EN,EM$,ET,ES
120 PRINTEN;EM$;ET;ES
125 RETURN
```

INDBUFFER consente di vedere quali indirizzi vengono assegnati ad ogni buffer nella RAM dell'unita' 1541.

```
5 REM INDBUFFER
10 DIMR$(8,7):FORK=1TO8:FORI=1TO7
15 R$(K,I)="":NEXTI:NEXTK
16 M1$="IND. IN MEMORIA DEL BUFFER: "
20 OPEN7,4:PRINT#7,"RISULTATI INDBUFFER"
25 OPEN15,8,15,"I"
30 INPUT"QUANTI BUFFER";N:PRINT#7
35 PRINT#7,N;" BUFFER":PRINT#7
40 A$="":B$="":C$="":D$="":E$=""
45 ONNGOTO70,65,60,55,50
50 OPEN2,8,2,"#":GOSUB270:GET#2,E$:GOSUB270
55 OPEN3,8,3,"#":GOSUB270:GET#3,D$:GOSUB270
60 OPEN4,8,4,"#":GOSUB270:GET#4,C$:GOSUB270
65 OPEN5,8,5,"#":GOSUB270:GET#5,B$:GOSUB270
70 OPEN6,8,6,"#":GOSUB270:GET#6,A$:GOSUB270
75 PRINT#7,"NUMERI DEI BUFFER: ";
80 ONNGOTO165,145,125,105,85
85 IFE$=""THENE$=CHR$(0)
90 PRINT#7,ASC(E$);
95 PRINT#15,"B-P:"2;1
100 PRINT#2,"22222";:GOSUB270
105 IFD$=""THEND$=CHR$(0)
110 PRINT#7,ASC(D$);
115 PRINT#15,"B-P:"3;1
120 PRINT#3,"33333";:GOSUB270
125 IFC$=""THENC$=CHR$(0)
130 PRINT#7,ASC(C$);
135 PRINT#15,"B-P:"4;1
140 PRINT#4,"44444";:GOSUB270
145 IFB$=""THENB$=CHR$(0)
150 PRINT#7,ASC(B$);
155 PRINT#15,"B-P:"5;1
160 PRINT#5,"55555";:GOSUB270
165 IFA$=""THENA$=CHR$(0)
170 PRINT#7,ASC(A$);
175 PRINT#15,"B-P:"6;1
180 PRINT#6,"66666";:GOSUB270
185 PRINT#7
190 FORK=1TO8:M=K-1
195 FORI=1TO7:L=I-1
200 PRINT#15,"M-R"CHR$(L)CHR$(M)
205 GET#15,R$(K,I)
210 NEXTI
215 NEXTK
220 FORK=1TO8:FORI=1TO7
225 IFR$(K,I)=""THENR$(K,I)=CHR$(0)
230 NEXTI:NEXTK
235 FORK=1TO8
240 PRINT#7,M1$;STR$((K-1)*256)
245 PRINT#7,"PRIMI 8 BYTE DEL BUFFER:";
250 FORI=1TO7
255 PRINT#7,R$(K,I);:NEXTI:PRINT#7
260 NEXTK
```

```
265 FORK=1TO7:CLOSEK:NEXTK:CLOSE15:STOP
270 INPUT#15,EN,EM$,ET,ES
275 PRINTEN;EM$;ET;ES
280 RETURN
```

CONTRINDI1 consente di vedere come il sistema assegna gli indirizzi di traccia e settore al file sequenziale INDI1, usato come indice degli archivi trattati nei precedenti paragrafi.

```
10 REM CONTRINDI1
20 OPEN15,8,15,"I"
30 OPEN2,8,2,"INDI1,S,R"
35 OPEN4,4:PRINT#4,"CONTROLLO INDI1":PRINT#4
40 K=0:I=0
45 INPUT#2,A$,T,S:TS=ST:K=K+1
50 PRINT#4,K;T;S;" ";
60 I=I+1:IFI=5THENPRINT#4:I=0
70 IFTS=64THENPRINT#4:CLOSE2:CLOSE4:CLOSE15:STOP
80 GOTO45
```

ALLOCA mostra come devono essere correttamente allocati i blocchi in modo diretto senza invadere la traccia 18, che viene usata per la directory.

```
1000 REM ALLOCA
1005 REM TRACCIA TX, SETTORE SX
1010 REM SI SUPPONE APERTO UN CANALE DATI
1015 REM SI SUPPONE APERTO IL CANALE 15
1020 REM SI SUPPONE CHE SX SIA COMPATIBILE CON T
X
1025 REM SCARTA LA TRACCIA 18
1030 PRINT#15,"B-A:0";TS;SX
1035 INPUT#15,EN,EM$,ET,ES
1040 IFEN=0THEN RETURN
1045 IFEN<>65THEN STOP
1050 IFET=18THENTX=19:SX=0:GOTO1030
1055 TX=ET:SX=ES:GOTO1030
```

CAPITOLO 7

ARCHITETTURA DEL SISTEMA

7.1 INTRODUZIONE

In questo capitolo affrontiamo un argomento avanzato; vogliamo descrivere il funzionamento del calcolatore nelle sue singole parti, cercando di renderlo comprensibile anche a chi non ha mai applicato CIRCUITI INTEGRATI, e non ha conoscenze di ELETTRONICA DIGITALE. Finora abbiamo visto il calcolatore come un apparecchio che, quasi per magia, comprende un linguaggio che assomiglia un po' alla lingua inglese, il BASIC. Qui trattiamo gli argomenti che occorre conoscere per essere in grado di adoperare il LINGUAGGIO MACCHINA, la cui conoscenza e' istruttiva ed assieme utile per risolvere problemi di velocita' e di gestione diretta delle periferiche.
Il tuo COMMODORE PLUS-4 possiede al suo interno diversi circuiti integrati; dei piu' importanti cercheremo di dare una descrizione soddisfacente, ma consapevolmente non completa per non disperdere in dettagli l'argomento.
Il COMMODORE PLUS-4, come tutti i calcolatori, e' costituito da circuiti integrati, collegati tra loro mediante connessioni elettriche. Ogni circuito integrato e' costruito in modo da svolgere determinate funzioni; il costruttore generalmente fornisce, a richiesta, la documentazione necessaria per applicarlo. Nel COMMODORE PLUS-4 alcuni integrati sono stati costruiti apposta dalla COMMODORE, e questa soluzione ha permesso di ridurre notevolmente il numero di componenti necessari per il funzionamento del calcolatore. Le principali componenti di un calcolatore sono tre:
1). UNITA' CENTRALE DI ELABORAZIONE (CPU, che in inglese significa CENTRAL PROCESSING UNIT)
2). MEMORIA
3). DISPOSITIVI DI INGRESSO/USCITA (I/O, cioe' INPUT/OUTPUT)
La potenza di un calcolatore dipende dalla potenza di queste tre parti.

7.2 LA MEMORIA

LA MEMORIA e' concettualmente la piu' semplice delle tre parti. Esistono nel COMMODORE PLUS-4 due tipi di memoria: a sola lettura (ROM, Read Only Memory) e a lettura e scrittura (RAM). Come dice il nome stesso, la MEMORIA e' un dispositivo che ha la caratteristica di ricordare, e difatti ricorda i bit, che sono l'unita' di informazione. Una cella (byte) di memoria e' l'insieme di 8 bit: nel COMMODORE PLUS-4 ci sono 65536 celle di memoria ROM e 65536 celle di memoria RAM.
La memoria ROM e' stata costruità con i bit a un valore prefissato, in modo tale da non essere alterabile nel tempo. La ROM contiene tutto il SISTEMA OPERATIVO, cioe' l'insieme delle routine in linguaggio macchina che occorrono per comunicare con le periferiche (tastiera, video, floppy disk, registratore, stampante, ecc), l'interprete BASIC, cioe' quel programma, scritto in linguaggio macchina, che, appoggiandosi alle routine del SISTEMA OPERATIVO, interpreta i comandi impartiti da tastiera, pone in memoria le linee introdotte, ed esegue le istruzioni memorizzate, e i 3+1 programmi applicativi.
La memoria RAM ha la caratteristica di essere alterabile: ogni byte di memoria RAM puo' cioe' essere scritto o letto, a seconda di un segnale, controllato dalla CPU (il segnale R/W, Read/Write). Quando questo segnale e' basso, l'accesso alla memoria viene effettuato in scrittura, altrimenti in lettura. La RAM e' usata per memorizzare i programmi dell'utente, le variabili, le informazioni da visualizzare sullo schermo ecc.
Nella Figura 7.1 e' illustrato il collegamento tra CPU e memoria. Per leggere un byte di memoria (vedi Figura 7.2) la CPU presenta sui 16 bit di indirizzo l'indirizzo del byte che vuole

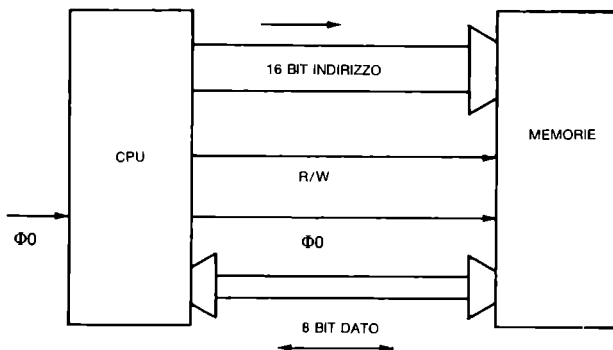



Figura 7.1 Collegamento tra CPU e memoria

leggere, pone a 1 il segnale R/W, attende una transizione del segnale di temporizzazione (CLOCK) e legge il dato dagli 8 bit dedicati ai dati.

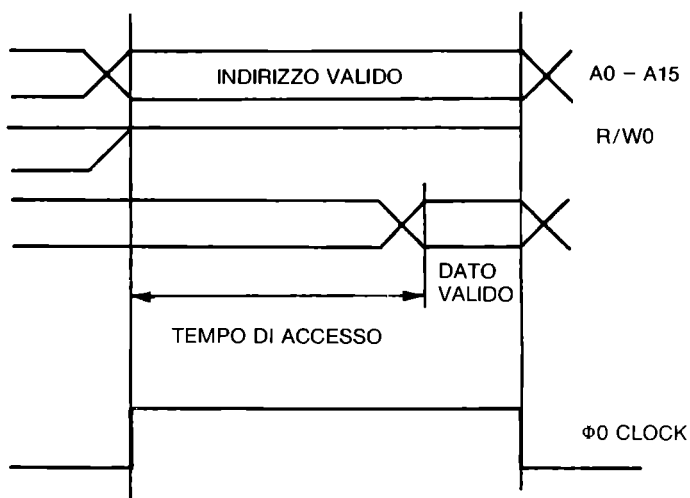



Figura 7.2 Ciclo di lettura da memoria

Per scrivere in memoria (vedi Figura 7.3) la CPU presenta sul DATA BUS (8 linee dedicate ai dati) il dato da scrivere, sul BUS INDIRIZZI (16 linee dedicate agli indirizzi) l'indirizzo del byte in cui memorizzare il dato e pone basso il segnale R/W. Dopo una transizione del segnale di CLOCK la CPU considera compiuta la scrittura.

7.3 L'UNITA' CENTRALE DI ELABORAZIONE (CPU)

La CPU e' considerata il CUORE del calcolatore, perche' esegue il programma, controlla i dispositivi di I/O e di fatto produce l'elaborazione dei dati secondo le istruzioni del programma. Nel COMMODORE PLUS-4 la CPU e' un circuito integrato con 40 piedini, cioe' 40 punti di collegamento con l'esterno. Questo integrato porta la sigla 7501, ed e' stato progettato apposta per i calcolatori COMMODORE 16 e PLUS-4. In realta' deriva da un'altra CPU,

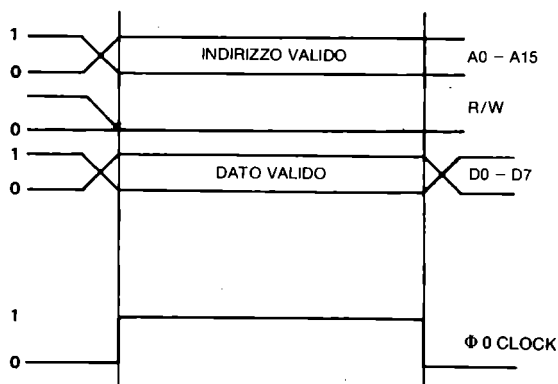



Figura 7.3 Ciclo di scrittura in memoria

la famosa 6502, da cui la 7501 differisce per alcuni dettagli, ma della quale conserva la stessa architettura e lo stesso set di istruzioni.
Prima di vedere l'architettura di questa CPU descriviamo brevemente i 40 segnali che vi fanno capo.

Segnale	Pin	Pin	Segnale
Φoin	1	40	RES
RDY	2	39	R/$\overline{W}$
$\overline{IRQ}$	3	38	D0
AEC	4	37	D1
Vcc	5	36	D2
A0	6	35	D3
A1	7	34	D4
A2	8	33	D5
A3	9	32	D6
A4	10	31	D7
A5	11	30	P0
A6	12	29	P1
A7	13	28	P2
A8	14	27	P3
A9	15	26	P4
A10	16	25	P6
A11	17	24	P7
A12	18	23	GATE IN
A13	19	22	A15
GND	20	21	A14

Figura 7.4 Zoccolatura della CPU 7501

Nella Figura 7.4 e' riportata la ZOCCOLATURA della CPU 7501.

. 1- Φ0 in. Ingresso. Da questo piedino la CPU riceve la temporizzazione (CLOCK) necessaria per il funzionamento. E' un' ONDA QUADRA della frequenza selezionabile di 890000 o 1780000 hertz. Tutte le operazioni della CPU sono legate al CLOCK: quando il CLOCK e' alto, la CPU accede a memoria, quando e' basso la CPU "pensa", cioe' esegue le commutazioni necessarie per eseguire l'istruzione, esegue calcoli, e non usa la memoria (vedi Figura 7.5).

.2- RDY. READY. Ingresso. Quando questo ingresso e' basso la CPU si arresta, e riparte quando RDY ritorna alto.

.3- IRQ. Interrupt ReQuest, ingresso. Quando si verifica una transizione da alto a basso di questo segnale la CPU "sente" una richiesta di interruzione, e va ad eseguire una speciale routine di servizio dell'interrupt (operazione simile alla chiamata e all'esecuzione di un sottoprogramma da programma, solo che qui la chiamata avviene per effetto di questo segnale), alla fine della quale il programma riprende.

.4- AEC. Address Enable Control, ingresso. Quando questo segnale viene posto basso la CPU non presenta gli indirizzi sul BUS INDIRIZZI. Cio' permette ad altri dispositivi di accedere alla memoria escludendo la CPU.

.5- VCC. Ingresso. Da questo piedino il circuito riceve l'alimentazione necessaria per funzionare. La tensione di alimentazione e' 5 volt in corrente continua, e la CPU assorbe circa 125 milliampere.

.6-19 e 21-22. ADDRESS BUS, uscite. Questi 16 piedini formano il BUS INDIRIZZI, cioe' la combinazione dei livelli logici di questi 16 bit individua uno tra i 2^16 (65536) possibili byte di memoria indirizzabili.

.20- GND, ingresso. E' il piedino di MASSA, da dove la CPU riceve il negativo di alimentazione, e rispetto al quale vanno misurati i livelli di tutti i segnali.

.23- GATE IN, ingresso.

.24-30. PO-P7, ingressi o uscite, programmabili. Questi 7 piedini fungono da I/O, e sono usati per collegare il calcolatore con il REGISTRATORE A CASSETTE, il FLOPPY DISK e la STAMPANTE.

.31-38 DATA BUS, ingressi e uscite. Su questi 8 BIT vengono scambiate le informazioni tra CPU e MEMORIA. Durante un ciclo di scrittura, sono uscite per la CPU, e ingressi per le memorie, mentre in un ciclo di lettura dalla memoria queste linee diventano uscite per la memoria e ingresso per la CPU.

.39- R/W, Read/Write, uscita. Questo segnale indica se il ciclo corrente di accesso alla memoria deve essere effettuato in lettura (livello ALTO) o in scrittura (livello BASSO).

.40 RES, ingresso, RESet. Questa linea e' sempre alta, durante il funzionamento del calcolatore. Quando viene posta bassa, la CPU arresta il programma in corso, e va ad eseguire la routine di

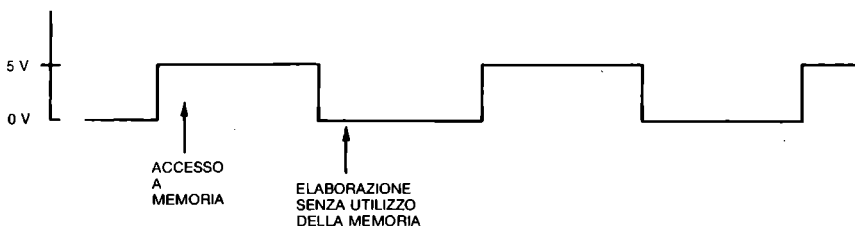



Figura 7.5 Segnale Φ0, il clock del sistema

inizializzazione. Il pulsante grigio sul fianco del COMMODORE PLUS-4 e' collegato alla linea di RESET. La linea di RESET viene posta bassa automaticamente all'accensione del calcolatore per circa un secondo.

7.4 I DISPOSITIVI DI INGRESSO/USCITA (I/O)

I dispositivi di INGRESSO/USCITA sono tutti quei dispositivi che collegano la CPU con l'esterno. La CPU piu' veloce del mondo, dotata della memoria piu' capace del mondo, sarebbe del tutto inutilizzabile se non fosse collegata ad almeno un dispositivo di ingresso e a un dispositivo di uscita. Immagina il tuo COMMODORE PLUS-4 privo di tastiera, video, suono, registratore a cassetta, stampante, floppy disk e di tutti gli altri dispositivi di I/O: non servirebbe a niente. Nei paragrafi che seguono non parliamo dei dispositivi fisici che sono o possono essere attaccati al calcolatore, ma vediamo come funziona il collegamento tra CPU e dispositivi. Premettiamo che la CPU 7501 non prevede un collegamento particolare con i dispositivi di I/O; per questo tali dispositivi vengono collegati in modo da apparire come celle di memoria. Una porta di I/O e' come una cella di memoria "aperta" (con i bit collegati a piedini), che puo' essere usata in ingresso o in uscita, il cui stato puo' essere letto (in ingresso) come un normale byte di memoria, o puo' apparire (in uscita) ad altre parti del calcolatore. Generalmente una porta di I/O ha 8 bit, cosi' come sono 8 i bit di un byte di memoria.

7.5 LA TASTIERA

Nella tastiera del COMMODORE PLUS-4 vi sono 67 tasti, ma poiche' i due tasti SHIFT e SHIFT LOCK, come anche i due CONTROL, sono collegati in parallelo, vi sono 64 tasti indipendenti. Si potrebbe pensare che vi siano 8 porte di I/O da 8 bit, cioe' che ognuno dei 64 tasti sia collegato a 1 bit di ingresso. Questa soluzione non sarebbe stata la piu' economica, e percio' la tastiera

e' stata organizzata in una matrice di 8 righe e 8 colonne, in cui i tasti collegano i fili orizzontali con quelli verticali (vedi Figura 7.6); una porta di uscita e' collegata alle righe, una porta di ingresso alle colonne.

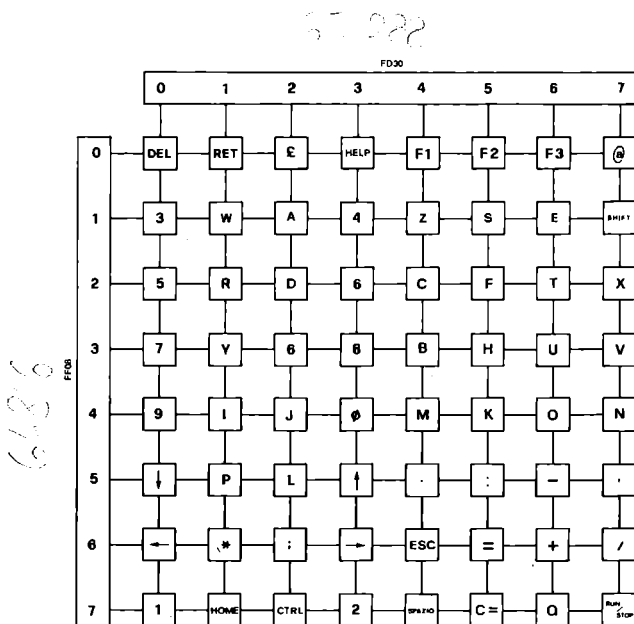



Figura 7.6 Organizzazione della tastiera

Per vedere se un tasto di una riga e' premuto, il SISTEMA OPERATIVO pone a 0 il bit della riga a cui quel tasto appartiene, a 1 i bit di tutte le altre righe, e legge dalla porta di ingresso lo stato delle colonne. Se qualcuno dei bit letti si trova a 0 allora vuol dire che il tasto corrispondente e' stato premuto, mentre gli ingressi delle colonne dove non sono premuti tasti vengono trovati nello stato logico 1. Il programma che segue, TASTMAT, si appoggia al SISTEMA OPERATIVO e mostra come la tastiera sia organizzata a matrice: la matrice disegnata ha 8 righe e 8 colonne: gli elementi della matrice sono quindi 64, e rappresentano ciascuno un tasto. Tenendone premuto uno vedi che un elemento della matrice cambia, e appare uno 0 al posto di un 1: nel programma infatti abbiamo rappresentato con 0 i tasti premuti e con 1 i tasti non premuti.

```
0 REM TASTMAT
10 SCNCLR:PRINT"   01234567":PRINT
20 DIMD(7):FORI=0TO7:D(I)=2↑I:NEXT
30 DO:FORI=0TO7
40 POKE2034,255-D(I):SYS56176:A=PEEK(2034)
50 PRINTI;:FORJ=0TO7
60 IFAANDD(J)THENPRINT"1";:ELSEPRINT"0";
70 NEXT:PRINT:NEXT
80 GETA$:A$=A$+" ":CHAR,16,5,A$
90 PRINTCHR$(19):PRINT
100 LOOP
```

COMMENTO A TASTMAT.

.10: Cancella lo schermo e stampa una riga

.20: Crea il vettore D e lo riempie con le potenze del due, da 2^0 a 2^7. E' un "trucco" per velocizzare il programma, poiche' il calcolatore e' molto piu' rapido nell'accedere ad un elemento di un vettore che non nel calcolare un'elevamento a potenza.

.30: Inizializza un ciclo DO e un ciclo FOR. Il ciclo DO ... LOOP e' eseguito all'infinito, e FOR ... NEXT e' ripetuto 8 volte, e ogni volta analizza lo stato di ciascuna delle 8 righe di cui e' composta la tastiera.

.40: Usa la routine del SISTEMA OPERATIVO che pone un byte nel registro di uscita, quello collegato alle 8 righe, e che ritorna lo stato del registro di ingresso, quello cioe' collegato alle 8 colonne. Il contenuto del byte 2034 viene caricato nell'accumulatore della CPU al momento dell'istruzione SYS, e alla fine della routine il contenuto dell'accumulatore viene posto nel byte 2037. La routine del SISTEMA OPERATIVO, che inizia all'indirizzo 56176, pone nel registro di uscita (righe) il contenuto dell'accumulatore (il valore posto nel byte 2034 prima di eseguire la SYS, e che contiene tutti i bit a 1, tranne il bit di posizione I), e torna con l'accumulatore che contiene il valore del registro di ingresso (colonne). Pone inoltre questo valore, che si trova ancora nel byte 2034, nella variabile A.

.50: Si prepara a stampare nella riga che sta analizzando, e inizializza un ciclo da 0 a 7 che usa per controllare lo stato degli 8 bit del registro di ingresso.

.60: Se il bit che analizza e' a 1, stampa 1, altrimenti stampa 0, una volta per ciascun bit di ciascuna riga.

.70: Chiude i due cicli FOR, e stampa un RETURN alla fine del primo ciclo.

.80: Stampa il carattere premuto, nella posizione 16,5. Vengono stampati correttamente solo i tasti che rappresentano un carattere alfanumerico, e non quelli di controllo o di funzione.

.90: Posiziona il cursore all'inizio della seconda riga: CHR$(19) infatti e' il carattere HOME.

.100: Chiude incondizionatamente il ciclo DO ... LOOP che parte dalla linea 30. Avremmo potuto usare GOTO 30, ma DO ... LOOP e' un'istruzione piu' elegante e piu' rapida da eseguire.

La scansione della tastiera viene effettuata automaticamente dal SISTEMA OPERATIVO ogni cinquantesimo di secondo, ed e' per questo che abbiamo fatto uso della routine in linguaggio macchina, anziche' effettuare da BASIC la scrittura nel registro di riga e la lettura dal registro di colonna: tra l'istruzione di scrittura e quella di lettura il SISTEMA OPERATIVO avrebbe effettuato probabilmente qualche scansione della tastiera, cambiando il contenuto del registro di riga. Terminiamo l'argomento indicando che il registro di uscita (quello che si usa per selezionare una riga) risponde all'indirizzo 64816 (FD30H), e quello di ingresso, che si usa per vedere a quale colonna appartiene l'eventuale tasto premuto, risponde all'indirizzo 65288 (FF08H). La modalita' di funzionamento di questo registro e' un po' particolare:

.occorre effettuare un'operazione di scrittura affinche' venga memorizzato lo stato delle linee esterne,

.effettuando dopo un ciclo di lettura e' possibile leggere lo stato delle linee esterne cosi' come era quando e' stato effettuato il ciclo di scrittura.

Questa complicazione e' stata necessaria per effettuare il collegamento con i joystick. Puoi verificare questa particolarita', se conosci gia' l'ASSEMBLER 6502, disassemblando la routine del SISTEMA OPERATIVO che noi richiamiamo nella linea 40 del programma TASTMAT, che parte dall'indirizzo 56176 (DB70H). Se non conosci ancora l'ASSEMBLER, puoi effettuare questa prova dopo aver letto il Capitolo 5.

7.6 I JOYSTICK

Al COMMODORE PLUS-4 si possono attaccare due joystick. Ogni joystick possiede 5 interruttori, 4 per le direzioni, alto, basso, sinistra, destra, e uno per il fuoco.

Nella Figura 7.7a e' riportato lo schema del collegamento con i joystick. Il piedino 8 dello spinotto del joystick 1 e' collegato al bit 2 del BUS DATI attraverso un BUFFER che serve solo per potenziare il segnale, prima di inviarlo all'esterno. Per leggere lo stato degli interruttori del joystick 1 e' sufficiente un ciclo di scrittura nel registro 65288 (FF08H), tale che il bit 2 del dato scritto sia a 0, e il bit 1 sia a 1. In questo modo se uno dei tasti del joystick 1 e' premuto, la linea collegata al piedino corrispondente a quel tasto risulta collegata al bit 2 del BUS DATI. Se si preme il bottone del fuoco il piedino 6 risulta collegato al bit 2. Poiche' siamo in un ciclo di scrittura, il bus dati contiene il valore del dato da scrivere, e in particolare il bit 2 contiene 0, come noi avevamo programmato. A questo punto il valore del registro di ingresso 65288 (FF08H) viene aggiornato in funzione dei suoi ingressi, e il bit 6 viene

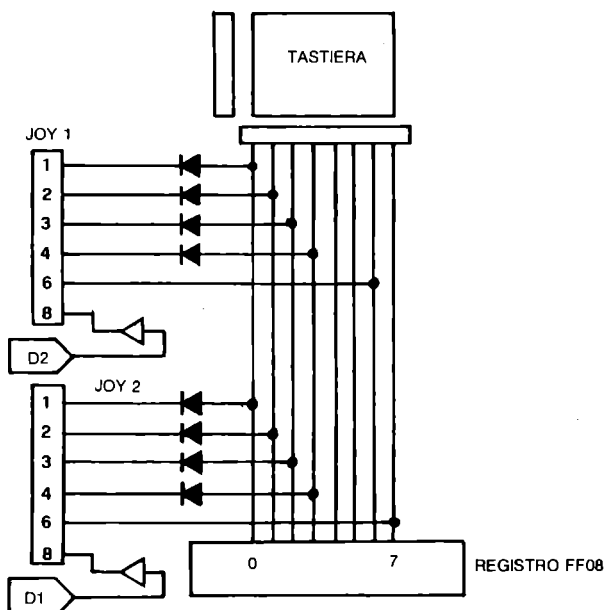



Figura 7.7a Collegamento con i Joystick

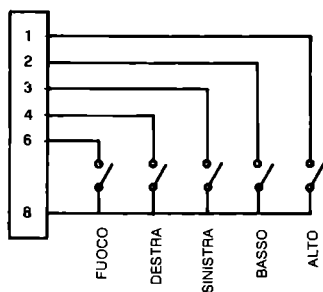



Figura 7.7b Schema elettrico dei Joystick

a trovarsi a 0, cosi' come e' a 0 il bit 2 del BUS DATI.
Nella Figura 7.7b e' riportato lo schema elettrico dei joystick. Le posizioni diagonali risultano dalla pressione contemporanea di due tasti, ad esempio "alto" piu' "sinistra" indicano che la leva del joystick e' posizionata in alto a sinistra. Nel Paragrafo

8.6 riportiamo una routine in linguaggio macchina che permette di leggere lo stato dei joystick.

7.7 IL REGISTRATORE A CASSETTE

Il registratore a cassette DATASSETTE 1531 si collega al COMMODORE PLUS-4 attraverso uno spinotto a 7 connessioni. Ecco il significato di questi 7 punti di connessione:

.1: MASSA. La massa del calcolatore va collegata con quella del registratore attraverso questa connessione.

.2: +5V. Il registratore ha bisogno di un'alimentazione a 5 volt per i circuiti elettronici di pilotaggio della testina, che riceve da questo punto.

.3: MOTORE. Da questo punto il motore del registratore riceve la corrente necessaria per il suo funzionamento, 9 volt in corrente continua non stabilizzati. Questa alimentazione puo' venire interrotta dal calcolatore, per arrestare il motore.

.4: LETTURA CASSETTA. Su questa linea vengono trasmessi i dati da registratore a CPU durante la lettura di dati dal nastro.

.5: SCRITTURA CASSETTA. Durante la scrittura di dati su nastro magnetico i dati provenienti dalla CPU vengono trasmessi al registratore attraverso questa linea.

.6: SENSORE. Questa linea e' collegata ai tasti del registratore, perche' la CPU sia in grado di sapere se viene premuto qualche tasto.

.7: MASSA. Anche questa linea e' collegata a massa, come la linea 1.

Ora che abbiamo visto il significato delle linee che effettuano fisicamente il collegamento, vediamo come queste linee possono essere gestite.
Il programma TASTREG, mostra come si possa leggere lo stato dei tasti del registratore: il bit 2 del byte di indirizzo 64784 (FD10H) non e' un bit di memoria, ma un bit di ingresso, il cui stato rispecchia lo stato dei tasti del registratore. Se tale bit e' a livello logico 1 nessuno dei 3 tasti PLAY, REWIND, FFWD e' premuto, altrimenti almeno uno di essi e' premuto.

```
0 REM TASTREG
10 DO
20 PRINT"PREMI UN TASTO SUL REGISTRATORE"
30 DO:LOOP WHILE PEEK(64784)AND4
40 PRINT"PREMI STOP SUL REGISTRATORE"
50 DO:LOOP UNTIL PEEK(64784)AND4
60 LOOP
```

COMMENTO A TASTREG
.10: Inizializza un ciclo DO ... LOOP infinito, per ripetere all'infinito il programma.
.20: Stampa un messaggio sul video, ad indicare che ora nessun tasto e' premuto.
.30: Attende che venga premuto un tasto sul registratore.
.40: Stampa un messaggio sul video, ad indicare che ora e' premuto almeno un tasto.
50: Attende che vengano rilasciati tutti i tasti del registratore.
60: Chiude il ciclo iniziato alla linea 10.

Abbiamo visto come scoprire se un tasto del registratore e' premuto oppure no. Il programma che segue, MOTREG, mostra come si possa arrestare o far partire il motore.

```
0 REM MOTREG
10 SCNCLR:PRINT"PREMI PLAY SUL REGISTRATORE"
20 PRINT:PRINT"A = MOTORE ACCESO"
30 PRINT"S = MOTORE SPENTO"
40 PRINT"F = FINE"
50 GOSUB120
60 GETKEYA$
70 IFA$="A"THENGOSUB120
80 IFA$="S"THENGOSUB150
90 IFA$<>"F"THEN60
100 GOSUB120
110 END
120 POKE0,PEEK(1)OR8
130 CHAR,0,6,"MOTORE ACCESO"
140 RETURN
150 POKE0,PEEK(1)AND247
160 CHAR,0,6,"MOTORE SPENTO"
170 RETURN
```

COMMENTO A MOTREG
.10/40: Stampa la videata con il menu'.
.50: All'inizio il motore viene acceso.
.60: Attende la pressione di un tasto sulla tastiera.
.70: Se il tasto premuto e' A accende il motore.
.80: Se il tasto premuto e' S spegne il motore.
.90: Se il tasto premuto non e' F torna alla linea 60.
.100: Prima di finire, accende il motore.
.110: Termina l'esecuzione del programma.
.120/140: Subroutine che accende il motore ponendo a 1 il bit 3 del byte 1 e stampa il messaggio "MOTORE ACCESO". Il motore e' collegato al bit 3 della porta di I/O inserita nella CPU 7501. Tale porta di I/O risponde all'indirizzo 1.

150/170: Subroutine che spegne il motore azzerando il bit 3 del byte 1 e stampa il messaggio "MOTORE SPENTO".
Dopo l'esecuzione del programma e' meglio eseguire un RESET del calcolatore.

Con questi due programmi abbiamo visto come si gestiscono dei dettagli sul registratore, i tasti e il motore, e non vediamo come effettivamente sono gestiti i dati. Il software di gestione della linea seriale del registratore e' troppo complesso per poter produrre riguardo ad esso degli esempi semplici: possiamo tuttavia illustrare i tipi di segnali che viaggiano sulle linee dati seriali da e verso il registratore. Le due linee LETTURA CASSETTA e SCRITTURA CASSETTA sono due linee digitali normali, e su di esse viaggiano segnali che possono assumere il valore 0 o 1: chi e' esperto di registrazioni magnetiche forse trova un po' strano questo fatto, dato che ci si sarebbe potuti aspettare un segnale sinusoidale, in cui una frequenza corrispondesse al livello logico 1 e un'altra frequenza al livello logico 0, con una modulazione in scrittura ed una demodulazione in lettura. Questo tipo di trasmissione, usato in diversi calcolatori, non viene usato nei calcolatori COMMODORE. Una semplice elettronica di pilotaggio nel registratore invia gli stessi livelli logici che riceve in ingresso alla testina, polarizzando in una direzione il nastro quando il livello logico e' alto, e nella direzione opposta quando il livello e' basso.

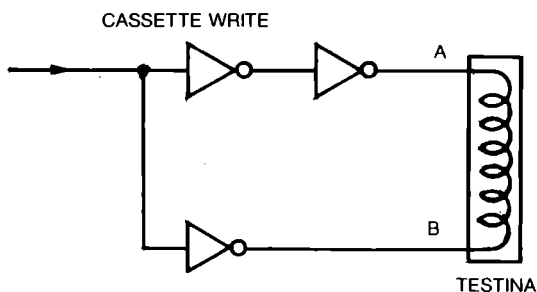



Figura 7.8a Collegamento della testina all'interno del registratore durante la fase RECORD

Nella Figura 7.8a e' riportata una semplificazione dello schema elettrico di pilotaggio della testina all'interno del registratore, durante la fase di registrazione, e nella Figura 7.8b sono illustrati i livelli di tensione applicati alla testina e la magnetizzazione corrispondente del nastro. Il tipo di colle-

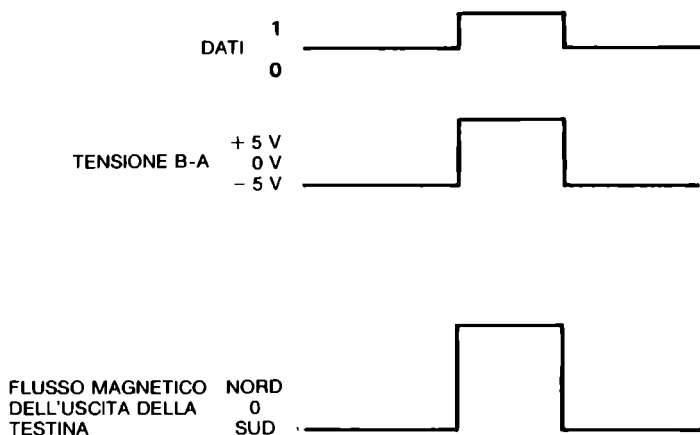



Figura 7.8b Livelli di tensione e magnetizzazione della testina del registratore

gamento adottato permette di avere all'uscita della testina (e quindi su nastro), sempre una polarizzazione magnetica, verso nord o verso sud, sia in corrispondenza del livello 0 che in corrisponaenza del livello 1. Se si fosse collegato il punto A o il punto B (vedi schema in Figura 7.8a) alla massa, in corrispondenza di uno dei livelli logici 0 o 1 ci sarebbe stata assenza di polarizzazione (anziche' polarizzazione in senso opposto), con conseguente instabilita' in fase di lettura dati. Il livello di registrazione risulta percio' essere sempre quello di saturazione del nastro, e per questa ragione conviene usare nastri ad alto livello di saturazione (HI ENERGY) per aumentare l'affidabilita' delle registrazioni.

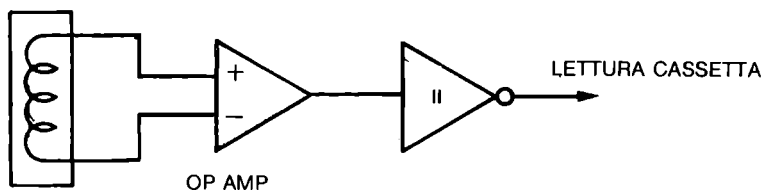



Figura 7.9 Collegamento della testina in fase PLAY

Nella Figura 7.9 riportiamo un'esemplificazione di come e' collegata la testina in fase di lettura: un amplificatore a guadagno elevato amplifica il segnale della testina e lo manda all'in-

gresso di un INVERTER che pilota la linea fino al registratore. La parte iniziale, l'ingresso dell'amplificatore, e' molto sensibile ai disturbi, per cui e' raccomandabile, soprattutto in fase di PLAY, tenere il registratore distante da fonti di disturbo, e in particolare distanti dallo schermo video, soprattutto se grande e a colori. La linea SCRITTURA CASSETTA e' collegata al bit 1 del byte 1, e la linea LETTURA CASSETTA e' collegata al bit 4 della porta 1. Il programma VEDEBIT stampa su video lo stato della linea LETTURA CASSETTA. Quando il registratore e' fermo o sulla cassetta non e' inciso segnale, il numero stampato su video non cambia, mentre durante l'ascolto di un pezzo di nastro registrato, vedi che a volte viene stampato 0, a volte 16 (2^4), a indicare che il bit 4 e' ora nello stato 0, ora nello stato 1. Non e' possibile in BASIC ricavare informazioni dal nastro, a causa della lentezza e della mancanza di temporizzazioni precise tipici del linguaggio.

```
0 REM VEDEBIT
10 PRINT"PREMI PLAY SUL REGISTRATORE"
20 DO:PRINTPEEK(1)AND16,:LOOP
```

COMMENTO A VEDIBIT
.10: Stampa il messaggio
.20: Stampa all'infinito il risultato della operazione logica AND tra 16 e il contenuto del byte 1, per considerarne solo il bit 4 (2^4=16). Nel Paragrafo 8.4.1 parliamo dettagliatamente di questa e di altre operazioni tra bit.

7.8 IL VIDEO

Il video e' diventato elemento essenziale nei calcolatori, poiche' da' una caratteristica di interattivita' assai gradevole ed utile sia nello scrivere che nell'eseguire i programmi. Vediamo come funziona uno schermo video: si tratta di uno schermo ricoperto di elementi (FOSFORI) che quando sono colpiti da elettroni emettono luce. Dietro questo schermo vi e' un "cannone" che emette un fascio di elettroni di intensita' regolabile verso il centro dello schermo. Poiche' gli elettroni sono carichi elettricamente, possono essere deviati da un campo elettrico. Per deviare gli elettroni vi sono delle placche sui 4 lati dello schermo: sopra, sotto, a destra e a sinistra. A seconda del potenziale applicato alle placche orizzontali si puo' orientare il fascio di elettroni verticalmente, mentre variando il potenziale applicato alle placche verticali lo si puo' orientare orizzontalmente. Per ottenere un'immagine sul video si e' diviso lo schermo in 312 righe orizzontali, e si fa passare il fascio di elettroni da sinistra verso destra dalla prima fino alla 312-esima riga in un 50-esimo di secondo. Un opportuno segnale regola poi l'intensita' del fascio che esce dal cannone, e cio' permette di avere punti diversamente luminosi sopra la stessa riga.

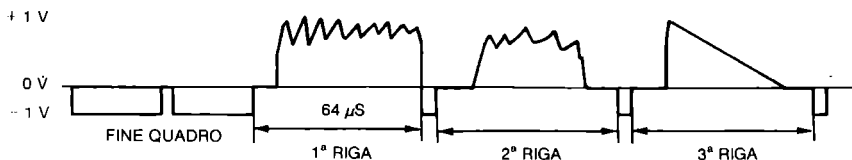



Figura 7.10 Segnale video per pilotare un monitor

Nella Figura 7.10 e' illustrato il formato del segnale video adatto a pilotare un normale monitor: la scansione di una riga dura 64 microsecondi. Il cannone "spara" elettroni in misura proporzionale al livello del segnale, istante per istante. Per disegnare una riga illuminata il segnale deve avere un livello di tensione alto per tutto il tempo in cui la riga deve essere illuminata, e il livello basso del segnale corrisponde al cannone che emette pochi elettroni, in modo da non far illuminare i fosfori che colpisce. I picchi di segnale negativi rappresentano i SINCRONISMI, e informano il monitor su quando andare a nuova riga, e quando andare a nuovo quadro. Il sincronismo di riga ha la frequenza di 15625 hertz, cioe' il periodo dura 64 microsecondi, e il sincronismo di quadro ha la frequenza di 50 hertz, cosicche' il fascio elettronico ritorna sullo stesso punto ogni 20 millisecondi.
Nel COMMODORE PLUS-4 c'e' un circuito integrato assai complesso che genera il segnale video. Tale integrato e' stato progettato appositamente dalla COMMODORE, porta la sigla 7360 ed e' stato battezzato TED, come TExt Display, perche' si occupa principalmente della visualizzazione del testo. Vediamo quali sono i compiti di TED e come li esegue: sappiamo che una pagina video di quelle che TED deve saper visualizzare e' formata da 40 X 25 caratteri. Poiche' ogni carattere occupa una matrice di 8 X 8 puntini (PIXEL) occorrono 320 punti su una riga. TED ha a disposizione circa 45 microsecondi (una parte della riga e' occupata dal bordo) per visualizzare una riga di testo, cioe' 140 nanosecondi (miliardesimi di secondo) per pixel. Come forse hai notato TED e' estremamente rapido. Ma vediamo come TED viene a conoscenza dei dati che deve visualizzare: tutte le informazioni di cui ha bisogno le puo' trovare nella memoria del calcolatore, la stessa che e' a disposizione della CPU. Per mostrare una riga TED ha bisogno di una quantita' rilevante di dati, in poco tempo:

. Il codice (D/CODE, vedi Appendice C) del carattere da visualizzare, 40 volte ogni otto righe.

. Il colore e la luminosita' del carattere, 40 volte ogni 8 righe.

. L'immagine del carattere da visualizzare, 40 volte ogni riga. Deduciamo che per visualizzare la prima di otto righe TED deve fare accesso a memoria ben 120 volte, mentre per le altre 7 righe bastano 40 accessi ciascuna, che vengono effettuati durante la fase bassa di Φ0 (il D/CODE del carattere e gli attributi sono gli stessi per 8 righe). Quando accede al codice e agli attributi TED disabilita la CPU; cio' comporta un rallentamento nell'esecuzione di tutti i programmi. Si puo' programmare TED in maniera che non disturbi la CPU, azzerando il bit 4 del registro $FF06 (65286 decimale), ma in questo caso non puo' visualizzare lo schermo, che appare tutto del colore del bordo. Ti ricordiamo che TED accede a memoria senza disabilitare la CPU durante la fase bassa di Φ0 (vedi Figura 7.5), ed utilizza questo tempo per leggere le immagini dei caratteri da visualizzare.

Nella Figura 7.11a sono elencate le locazioni di memoria che controllano la disposizione dei caratteri sullo schermo e nella Figura 7.11b le locazioni che controllano gli attributi, cioe' il colore, il livello di luminosita' e il lampeggiamento dei caratteri. Sono queste, oltre all'immagine dei caratteri, residente in ROM, le zone di memoria da cui il TED attinge le informazioni necessarie per effettuare la visualizzazione dello schermo nel modo testo. La memoria degli attributi contiene i dati relativi alla luminosita' (i bit 6-5-4), al colore (i bit 3-2-1-0) e al lampeggiamento (il bit 7) di ciascun carattere. Gli indirizzi segnati nella figura sono quelli validi all'accensione del calcolatore, e vedrai nel Capitolo 9 come si possono modificare. La programmazione del TED e' assai laboriosa, soprattutto a causa del gran numero di funzioni che puo' eseguire, e per questo affrontiamo l'argomento in diversi punti, particolarmente nel Capitolo 9 dedicato alla grafica.

TED ha un numero di funzioni assai elevato all'interno del COMMODORE PLUS-4: genera il CLOCK per la CPU, genera il segnale video, genera il segnale audio, rinfresca le memorie dinamiche e regola le temporizzazioni necessarie per la gestione di tali memorie, riceve i dati che provengono dalla tastiera e quelli che provengono dai joystick, genera gli INTERRUPT per la CPU, emette i segnali di abilitazione per le ROM, e infine seleziona il banco ROM o RAM negli indirizzi 8000H-FFFFH. Come vedi e' riduttivo pensare al TED solo come interfaccia per il video: il TED e' un integrato che effettua gran parte delle funzioni di I/O e di interfacciamento con la memoria.

Il segnale video esce dal calcolatore dalla presa marcata VIDEO, presente sul retro della tastiera. Il piedino 1 emette un segnale che contiene LUMINANZA e SINCRONISMI, ma non contiene le informazioni relative al colore, che escono invece dal piedino 6 (CROMINANZA). Il piedino 4 contiene un segnale VIDEO COMPOSTO, che comprende LUMINANZA, CROMINANZA e SINCRONISMI. Al piedino 1 si puo' attaccare un monitor monocromatico; il monitor a colori

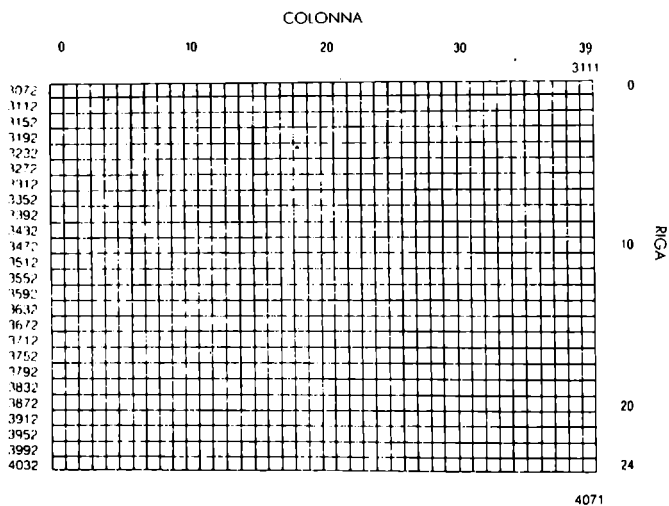



Figura 7.11a Mappa della memoria dei caratteri

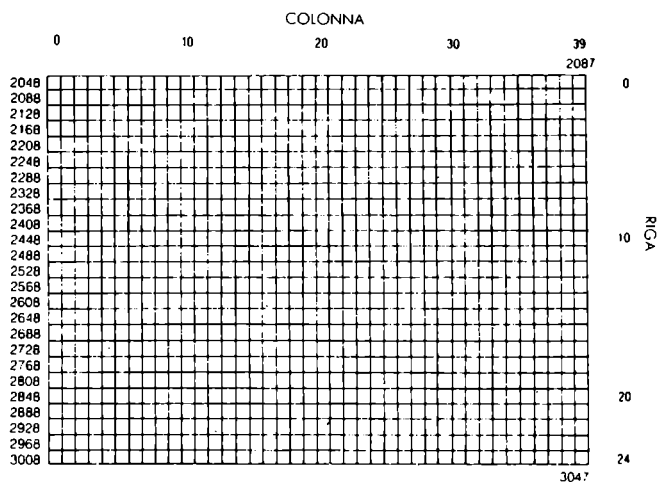



Figura 7.11b Mappa della memoria del colore e della luminosita' dei caratteri

della COMMODORE modello 1701/1702 fa uso dei segnali separati LUMINANZA+SINCRONISMI e CROMINANZA, che preleva dai piedini 1 e 4 dello spinotto, mentre i monitor di altre marche generalmente fanno uso del segnale VIDEO COMPOSTO presente al piedino 7. La qualita' del segnale VIDEO COMPOSTO e' inferiore a quella dei segnali separati LUMINANZA e CROMINANZA, ed e' anche per questo che la qualita' dell'immagine su monitor COMMODORE 1701 e' migliore di quella di quasi tutti gli altri monitor della stessa fascia di prezzo. Il segnale per il video esce anche da un altro spinotto, quello marcato RF, modulato con radiofrequenza. Tale segnale e' adatto all'uso con un normale televisore, ma la qualita' dell'immagine e' generalmente peggiore di quella del monitor. Il televisore va sintonizzato sul canale 36, in banda IV (UHF), e talvolta puo' essere necessario ritoccare la sintonia del TV a causa di inevitabili slittamenti della frequenza che dipendono dalla variazione di temperatura dei componenti del modulatore. La Figura 7.12 riporta lo schema di collegamento dello spinotto VIDEO.

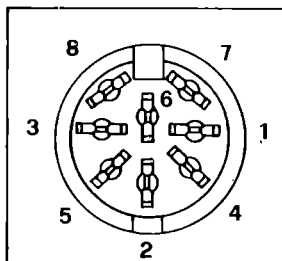
VISTA FRONTALE
DELLA PRESA (FEMMINA)

PIEDINO	FUNZIONE
1	LUMINANZA + SINCRONISMI
2	MASSA
3	USCITA AUDIO
4	VIDEO COMPOSTO
5	AUDIO IN
6	CROMINANZA
7	NON COLLEGATO
8	+ 5 VOLT

Figura 7.12 Connessioni della presa AUDIO/VIDEO

7.9 IL SUONO

Il suono nel COMMODORE PLUS-4 viene generato da due oscillatori che si trovano nel TED. Per ognuno di questi vi e' un registro a 10 bit che indica la frequenza di oscillazione; vi e' poi un registro comune, di quattro bit, che indica il volume del suono emesso: il livello 0 corrisponde a silenzio, e il livello 8 al massimo (i livelli da 8 a 15 sono equivalenti). Tre bit servono ad azionare le due voci, e uno di questi indica se la voce 2 emette un segnale a onda casuale (rumore bianco).

FREQ.: VOCE 1	9 8	7 6 5 4 3 2 1 0
	BIT 1 E 0 $ FF12	F F 0 F
FREQ.: VOCE 2	9 8	7 6 5 4 3 2 1 0
	BIT 1 E 0 $ FF.10	F F 0 F

ABILITAZIONE: BIT 4 $ FF11 (0 = OFF)
VOCE 1

ABILITAZIONE: BIT 5 $ FF11 (0 = OFF)
VOCE 2

ABILITAZIONE:
RUMORE BIANCO: BIT 6 $ FF11 (0 = OFF)
PER VOCE 2

DISABILITAZIONE: BIT 7 $ FF11 (1 = DISABILITATO)
DEL SUONO

VOLUME: BIT 03 $ FF11

Figura 7.13 Registri per la gestione del suono

Nella Figura 7.13 sono riportati gli indirizzi e le posizioni di tutti i bit che riguardano il suono. Per generare rumore bianco e' sufficiente selezionare il bit del rumore bianco, e non e' necessario selezionare anche il bit di abilitazione della voce 2, dopo aver impostato la frequenza.
Il programma SUONOCONPOKE e' un esempio di come vanno gestiti i registri di TED per generare suoni.

```
0 REM SUONOCONPOKE
10 T=255*256
20 POKET+17,8+32
30 FORF=1TO4*256-1STEP10
40 POKET+15,FAND255:POKET+16,F/256:NEXT
50 POKET+15,200:POKET+16,0
60 FORV=8TO0STEP-1:POKET+17,V+32
70 FORR=1TO200:NEXT:NEXT
```

Il suono esce dallo spinotto marcato VIDEO presente sul retro della tastiera, ed e' collegato al piedino 3. Per chi usa il televisore il segnale del suono e' modulato, e appare nell'altoparlante del TV come quello di un normale canale televisivo. Nella Figura 7.12 e' riportato lo schema dello spinotto marcato VIDEO. Il piedino 5 dello spinotto, AUDIO IN, permette di inserire un segnale audio in ingresso: il segnale audio in uscita AUDIO OUT comprende il mixaggio di AUDIO IN e del suono generato dal TED.

7.10 LA PORTA SERIALE

Se per i calcolatori stai acquistando quella sorta di passione che ha conquistato ed anima noi che scriviamo, ti troverai, prima o poi, nella necessita' di espandere il tuo calcolatore con il DISK DRIVE e con la STAMPANTE: potrai usare il tuo sistema calcolatore per gestire archivi elettronici, per scrivere lettere e libri (questo libro e' stato scritto su COMMODORE 64, con il programma di elaborazione testi EASY SCRIPT, e stampato con una stampante a margherita: EASY SCRIPT sara' presto disponibile anche per COMMODORE PLUS-4), e per automatizzare quelle operazioni che riterrai possibile e utile automatizzare. Il tuo COMMODORE PLUS-4 potra' ancora esserti utile, poiche' le sue doti di espandibilita' e i programmi che gradualmente vengono presentati sul mercato ti permetteranno di usare questo calcolatore anche per usi "professionali". La porta seriale permette di collegare FLOPPY DISK DRIVE e STAMPANTE, oltre ad eventuali altre apparecchiature (PLOTTER, ecc) che ora o in seguito potranno essere costruite.

CONNETTORE

Pin No.	Signal
1	N.C.
2	GND
3	SERIAL ATN
4	SERIAL CLK
5	SERIAL DATA
6	RESET

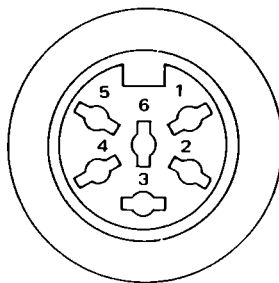



Figura 7.14 Schema della porta di I/O seriale (SERIAL BUS)

Vediamo nella Figura 7.14 lo schema di collegamento della porta di I/O seriale (SERIAL BUS). I nomi dei segnali sono in inglese, poiche' nella terminologia corrente se ne fa uso in lingua inglese.

Il collegamento tra calcolatore e periferiche su SERIAL BUS e' abbastanza complesso, tuttavia possiamo illustrare il principio logico di funzionamento, e il significato dei segnali che realizzano il collegamento. Il SERIAL BUS e' stato progettato in maniera da permettere il collegamento con un numero e un tipo variabile di periferiche. Se osservi attentamente la Figura 7.14 ti accorgi che i fili su cui effettivamente viaggiano segnali utili per lo scambio delle informazioni sono 3, piu' la massa. Attraverso questi 4 fili e' possibile comunicare con un numero di periferiche che puo' arrivare anche a 7 (2 STAMPANTI, numero 4 e 5, un PLOTTER, numero 6, quattro DISK DRIVE, numeri da 8 a 11). Appare evidente come vi debba essere un accorgimento per comunicare con 7 diverse unita' attraverso 4 soli fili. Poniamo l'attenzione sul SERIAL BUS, cioe' su questi 4 fili a cui risultano collegati, per fare un esempio: CALCOLATORE, DISK DRIVE e STAMPANTE.

E' necessario che i dati siano in grado di viaggiare nelle seguenti direzioni:

.da calcolatore a stampante,
.da calcolatore a drive,
.da drive a calcolatore.

Non e' previsto un collegamento in cui il calcolatore sia escluso, cioe' da drive a stampante. E' stato stabilito che sul SERIAL BUS possono essere collegati tre tipi di dispositivi:

- CONTROLLER, cioe' il dispositivo che controlla il flusso dei dati; chi assume questa funzione e' sempre il calcolatore,
- TALKER, cioe' il dispositivo che invia i dati,
- LISTENER, cioe' il dispositivo che riceve i dati.

Inoltre un dispositivo puo' essere in uno stato in cui ignora i

dati che viaggiano sul bus.
Per effettuare la trasmissione dei dati il controller invia dei messaggi agli apparecchi collegati, e attende la loro risposta. Ogni apparecchio ha un numero che lo identifica: la STAMPANTE, ad esempio, normalmente ha il numero 7. Per comunicare con la stampante il calcolatore invia all'unita' 4 l'ordine di ascoltare (diventare LISTENER). Se l'unita' 4 risponde entro un ventesimo di secondo, il calcolatore invia i dati e la stampante li stampa, altrimenti viene rilevato un errore di mancanza del dispositivo (DEVICE NOT PRESENT). Alla fine di ogni dato la stampante invia una risposta, per informare il calcolatore che il dato e' stato accettato, per evitare che si perdano dei dati a causa del fatto che la stampante e' piu' lenta a stampare di quanto lo sia il calcolatore ad inviare i caratteri. Alla fine della stampa il calcolatore comanda alla stampante di uscire dallo stato di LISTENER, e di ignorare gli eventuali caratteri futuri (potranno anche non essere indirizzati a lei).

Il software per il collegamento con il DRIVE e' un po' piu' complesso, perche' il DRIVE, a differenza della stampante, e' in grado sia di ricevere, sia di trasmettere dati: il modo di ricezione dei dati e' uguale a quello della stampante, mentre la trasmissione da DRIVE a CALCOLATORE avviene nel modo seguente: il calcolatore, che ha bisogno di informazioni dal drive, trasmette al drive i dati di cui ha bisogno (ad esempio, il nome del file che vuole caricare in memoria), dopodiche' comanda al drive di diventare TALKER (colui che parla). Il drive invia tutti i caratteri del file (il calcolatore intanto ascolta), e alla fine trasmette una sequenza di FINE TRASMISSIONE. Il calcolatore comanda percio' al drive di non trasmettere e non ascoltare piu' i dati, in modo da liberare la linea seriale per i collegamenti anche con le altre periferiche.
Come forse hai gia' notato, il calcolatore e' in grado di comandare alle unita' presenti su linea seriale di ascoltare, di parlare, di ignorare. Tutte queste informazioni speciali possono partire solo dal calcolatore, e sono segnali trasmessi sotto ATTENTION. Il SERIAL ATN OUT e' un segnale che esce dal calcolatore, e significa che i dati che viaggiano sulle linee CLOCK e DATA sono da interpretarsi come comandi. In questo modo il calcolatore trasmette due byte da otto bit: il primo contiene il numero di periferica a cui e' inviato il comando (4 bit meno significativi) e il tipo di comando da eseguire (LISTEN, TALK, UNLISTEN, UNTALK, 4 bit piu' significativi); il secondo byte contiene un numero chiamato INDIRIZZO SECONDARIO che la periferica riceve, ed ha un significato che dipende dalla periferica a cui e diretto. Quando e' attivo il segnale ATTENTION nessuna periferica presenta dati sulle linee SERIAL CLOCK e SERIAL DATA, in modo da permettere al calcolatore di eseguire la sequenza di ATTENTION, e tutti i dispositivi ascoltano, pronti a eseguire i comandi invia-

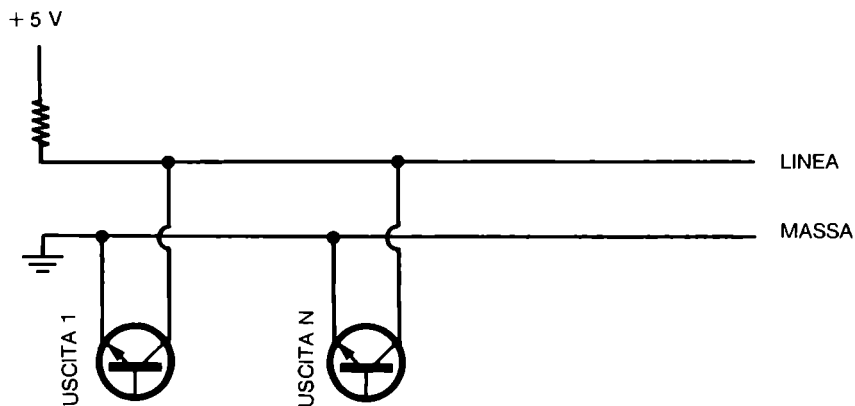



Figura 7.15 Schema elettrico del SERIAL BUS

ti. Come puoi osservare nella Figura 7.15, tutte le linee sono di tipo OPEN COLLECTOR, cioe' l'uscita di ogni dispositivo collegato e' in grado di porre la linea BASSA se la linea e' alta, ma nessuna uscita e' in grado di riportare la linea ALTA, se qualche altra unita' la tiene bassa.
E' chiaro che la linea e' libera quando nessun dispositivo la pone bassa, cioe' quando tutte le uscite sono alte. Se non conosci il funzionamento di un transistor, puoi immaginare i transistor disegnati in corrispondenza di ciascuna uscita come degli interruttori: se l'interruttore e' chiuso la linea si trova collegata alla massa, e il livello e' BASSO; se l'interruttore e' aperto lo stato della linea puo' essere basso o alto, e dipende dagli altri interruttori: se tutti gli interruttori sono aperti, allora la linea assume il livello alto, a causa del resistore collegato al positivo di alimentazione.

7.11 L'ORGANIZZAZIONE DELLA MEMORIA

Nella Figura 7.16 e' illustrata l'organizzazione della memoria nel COMMODORE PLUS-4. Abbiamo visto nei paragrafi precedenti che questo calcolatore e' dotato di 64K BYTE di memoria RAM.

Considerando il fatto che la CPU dispone di soli 16 fili per indirizzare tutte le memorie collegate (ROM, RAM, I/O), puo' cioe' indirizzare al massimo 64K di memoria, e' lecito chiedersi dove siano i 64K di ROM, e dove siano i registri di I/O. Guardando la Figura 7.16 puoi osservare come gli indirizzi di I/O

siano comuni a tutte le mappe, partendo da $FD00 fino a $FF40. 32K di ROM possono alternarsi alla RAM negli indirizzi da $8000 a $FFFF (escluso l'I/O). Per abilitare la ROM a rispondere agli indirizzi suddetti si deve effettuare un'operazione di scrittura nel registro del TED che risponde all'indirizzo $FF3E, mentre per abilitare la RAM basta effettuare la scrittura nel registro di indirizzo $FF3F. Per selezionare quali ROM utilizzare, se quelle del S.O. e BASIC o quelle dei 3+1, devi effettuare una scrittura in $FDD0 per le prime o in $FDD5 per le seconde.

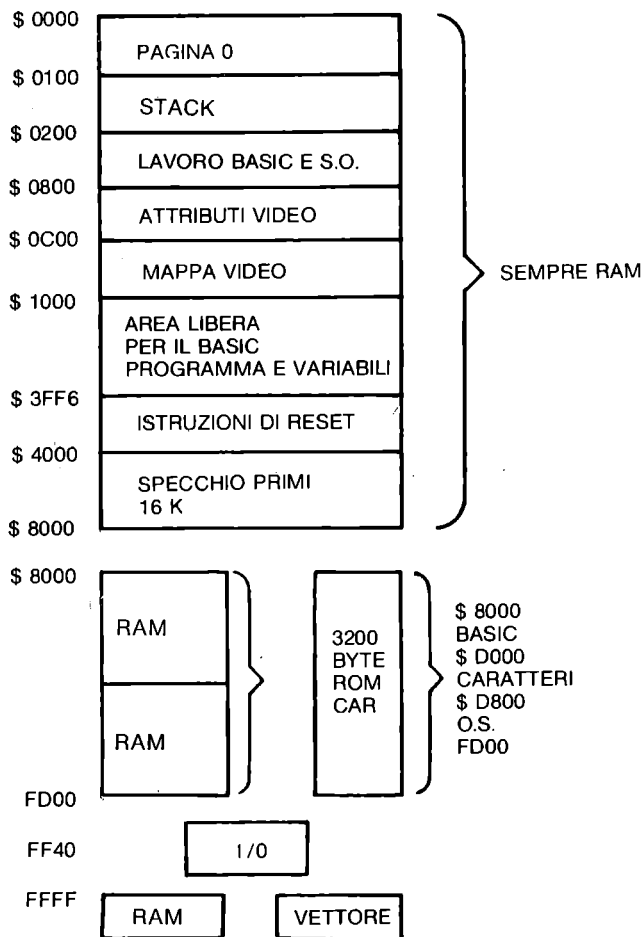



Figura 7.16 Organizzazione della memoria

CAPITOLO 8

LA PROGRAMMAZIONE IN ASSEMBLER E IN LINGUAGGIO MACCHINA

8.1 INTRODUZIONE

Nel Capitolo 7 abbiamo esposto le caratteristiche HARDWARE del COMMODORE PLUS-4 anche per metterti in condizione di utilizzare i dispositivi che lo compongono programmando in ASSEMBLER. Uno degli ostacoli piu' grossi nell'apprendimento della programmazione in linguaggio macchina o in ASSEMBLER e' quello di far comunicare il calcolatore con l'esterno. In questo capitolo vediamo come e' fatta la CPU 7501, quali istruzioni e' in grado di eseguire, come e' fatto e come viene eseguito il programma.
Il programma che la CPU esegue e' contenuto in memoria come una sequenza di byte. La CPU preleva dalla memoria il byte, che rappresenta l'istruzione da eseguire, e gli eventuali operandi indicati nei byte seguenti; esegue l'istruzione e passa a prelevare l'istruzione successiva. Ogni istruzione e' quindi letta dalla memoria come un normale byte. Programmare in linguaggio macchina vuol dire porre nei byte di memoria i codici delle istruzioni che si vogliono eseguire, e gli operandi. Come puoi immaginare, e' assai laborioso scrivere un programma sotto forma di numeri (in binario), e per questo non e' quasi mai usata questa tecnica.
Il linguaggio macchina lavora su dati numerici contenuti in un byte, quindi con valore da 0 a 255. Per poter trattare numeri piu' grandi o con segno si devono usare sottoprogrammi adatti. Lo stesso discorso vale per trattare numeri con cifre dopo il punto decimale (floating-point) e dati alfanumerici.
Le istruzioni che la CPU riconosce hanno ricevuto un nome mnemonico che si preferisce usare per scrivere il programma. Esistono dei programmi (alcuni chiamati ASSEMBLATORI), che traducono in codice oggetto (combinazione di zeri e uni da porre in memoria) il programma redatto da noi in forma mnemonica (che viene chiamato programma sorgente). Il comando MONITOR del BASIC 3.5 pone a nostra disposizione proprio un programma traduttore, anche se non si tratta di un vero assemblatore. Parliamo diffusamente del MONITOR nel Paragrafo 8.6.

Anche se, grazie al MONITOR, non dobbiamo tradurre in numeri i codici delle istruzioni, abbiamo a che fare con i byte della memoria per gli operandi delle istruzioni e per gli indirizzi. Il sistema di numerazione decimale non consente una conversione immediata in binario. D'altra parte il sistema binario richiede troppe cifre per indicare il contenuto di un byte: ad esempio il numero 221 decimale corrisponde al numero binario 11011101. Il sistema di numerazione che e' stato scelto per indicare gli operandi delle istruzioni e' quello a base 16, o ESADECIMALE. In tale sistema vi sono 16 cifre: 0 1 2 3 4 5 6 7 8 9 A B C D E F. Il vantaggio dell'esadecimale rispetto al decimale e' che un numero esadecimale si puo' tradurre in binario piu' in fretta, perche' ogni cifra rappresenta 4 bit; rispetto al binario il vantaggio e' che i numeri sono piu' compatti. In questo libro e nei libri di ASSEMBLER 6502 i numeri esadecimali sono preceduti dal segno dollaro: il numero $23 si rappresenta in binario 00100011.

ESADECIMALE	BINARI	DECIMALE
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Figura 8.1 Esadecimale, binario e decimale

Nella Figura 8.1 trovi la corrispondenza dei valori delle cifre esadecimali nei sistemi binario e decimale. Come nel sistema decimale dopo il numero 9 si passa à 10 ponendo 1 nella posizione piu' a sinistra, cosi' nell'esadecimale dopo $F si passa a $10, che vuol dire 16 in decimale. La Figura 8.2 mostra ancora un esempio di conversione: lo stesso numero 101 viene analizzato

prima come binario, poi come esadecimale e poi come decimale. A titolo informativo aggiungiamo che nel mondo dei calcolatori si fa spesso uso anche del sistema a base 8, detto OTTALE, dove le cifre vanno da 0 a 7. Noi non faremo mai riferimento a tale sistema in questo libro, e per questo non lo approfondiamo.

Figura 8.2 Analisi del numero 101

Accenniamo, da ultimo, al sistema BCD (Bynary Coded Decimal), cioe' decimale a codifica binaria: e' una via di mezzo tra esadecimale e decimale, perche' ogni cifra BCD rappresenta 4 bit (come l'esadecimale) e le cifre vanno da 0 a 9 (come il decimale). Il difetto di questo sistema di rappresentazione e' un notevole spreco di bit: un byte puo' infatti contenere solo 100 combinazioni ammesse (da 0 a 99), invece di 255 possibili in binario. Un altro svantaggio risiede nell'elaborazione: consideriamo come esempio una qualsiasi operazione di somma:

```
NUMERO   RAPPRESENTAZIONE BCD
  08 +        00001000 +
  12 =        00010010 =
------       ----------
  20          00100000 invece di 00011010
                       come deve essere.
```

Osserviamo subito che l'unita' di calcolo all'interno della CPU deve lavorare diversamente se gli operandi e il risultato sono in BCD da come opera con gli operandi in binario. Nella 7501 e' possibile selezionare il modo BCD con un'istruzione, e tornare al modo binario con un'altra istruzione, e questa possibilita' aiuta notevolmente il programmatore ASSEMBLER nel gestire i dati decimali.

8.2 ORGANIZZAZIONE DELLA CPU 7501

La 7501 e' una CPU che deriva dalla famiglia 6500, una serie di unita' centrali di elaborazione sviluppata negli anni '70 negli USA dalla ROCKWELL INTERNATIONAL CORPORATION. La "filosofia" dei progettisti e' stata quella di creare una CPU dotata di un numero di istruzioni relativamente limitato, un numero di registri interni anch'esso limitato, ma di potenti e veloci modi di indirizzamento. Per limitare il numero di istruzioni non si sono poste istruzioni specifiche per gestire l'I/O: la CPU gestisce solo la memoria, e l'I/O deve essere collegato alla CPU come la memoria. Il limitato numero di registri e' compensato dalla gestione rapida dei dati in PAGINA ZERO: le celle di memoria che partono dall'indirizzo 2 fino a $00FF sono da considerarsi come un'estensione dei registri interni della CPU, poiche' si possono indirizzare piu' rapidamente e permettono indirizzamenti assai potenti, come vedremo nel prossimo paragrafo.

Osserviamo la Figura 8.3 per vedere come funziona internamente la CPU. Partiamo dai registri: la 7501 contiene 6 registri a disposizione del programmatore. Ogni registro e' formato da 8 bit (tranne il PROGRAM COUNTER che ha 16 bit), ed ha la caratteristica di poter immagazzinare o fornire alla propria uscita un dato di 8 bit. I registri non elaborano le informazioni, ma semplicemente "ricordano" lo stato degli 8 bit di cui sono composti. I registri sono i seguenti:

. Index register Y: REGISTRO INDICE Y
. Index register X: REGISTRO INDICE X
. Stack pointer register: STACK POINTER
. Accumulator: ACCUMULATORE
. PCL: meta' del PROGRAM COUNTER, gli 8 bit meno significativi
. PCH: meta' del PROGRAM COUNTER, gli 8 bit piu' significativi
. Processor status register: registro che contiene 7 bit; ognuno di questi bit, chiamati in gergo FLAG, ha un significato che riflette lo stato interno della CPU; esso dipende dalle istruzioni eseguite o dal risultato dell'ultima operazione. Dei FLAG parleremo piu' dettagliatamente nel seguito di questo paragrafo.

Tutti i registri sono collegati ad un BUS DATI interno, e attraverso questo BUS (insieme di 8 fili che corrispondono ciascuno a un bit) possono caricare o scaricare dati. Ogni registro risulta collegato anche a una parte della CPU indicata nello schema come INSTRUCTION DECODE (decodifica dell'istruzione): questo blocco contiene l'unita' di controllo della CPU, unita' che controlla tutte le abilitazioni dei registri (puo' comandare a un registro di caricare il dato presente sul BUS DATI interno, o di presen-

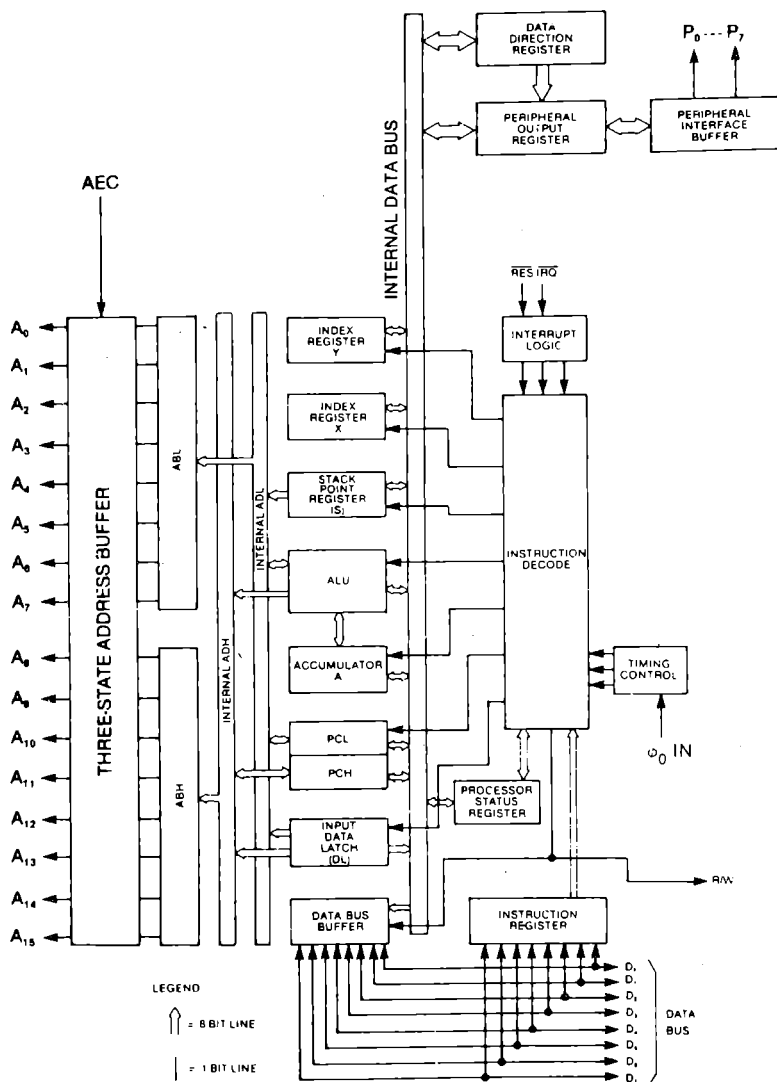



Figura 8.3 Architettura interna della CPU 7501

tare su tale BUS il proprio contenuto) e l'ALU (unita' aritmetico-logica). La decodifica dell'istruzione e' la parte piu' complessa di tutta la CPU: la sua funzione e' quella di generare una sequenza di abilitazioni dei registri in modo da eseguire l'istruzione specificata dal programma (il codice dell'istruzione da eseguire viene posto nell'INSTRUCTION REGISTER, che lo presenta al blocco INSTRUCTION DECODE, come puoi osservare nella parte in basso a destra dello schema).
Abbiamo nominato l'ALU; puoi osservare nello schema che tale unita' e' collegata al BUS DATI interno, all'ACCUMULATORE, e ai due BUS interni ADL e ADH, che sono la parte bassa (ADL, 8 bit meno significativi) e parte alta (ADH, 8 bit piu' significativi) del BUS INDIRIZZI interno. L'ALU e' in grado di eseguire operazioni aritmetico-logiche tra i vari BUS a cui e' collegata. Ad esempio l'ALU e' in grado di eseguire la somma tra il contenuto del BUS DATI interno e l'accumulatore, e memorizzare il risultato. Essa e' in grado poi di presentare il risultato memorizzato sul bus dati; tutte le operazioni che l'ALU esegue sono comandate dalla sezione INSTRUCTION DECODE.
Vediamo ora come la CPU esegue un'istruzione, ad esempio l'istruzione:
LDA #$24
che si trova nei byte di memoria $1000 e $1001:

```
$1000 = $A9
$1001 = $24
```

L'istruzione LDA # produce l'effetto di caricare nell'accumulatore il dato che segue. Il codice di questa istruzione e' 10101001 binario, cioe' $A9.
Il PROGRAM COUNTER (che da ora chiameremo P.C.) contiene l'indirizzo dell'istruzione che deve essere eseguita, cioe' nel nostro caso $1000. Il contenuto del P.C. viene presentato sul BUS INDIRIZZI, si attende una transizione di Φ0 in modo che la memoria abbia il tempo di fornire il dato richiesto, e si carica il contenuto del BUS DATI esterno (in basso nel disegno) nel registro delle istruzioni. Nel nostro caso tale registro contiene il contenuto del byte $1000, cioe' $A9. INSTRUCTION DECODE (da ora I.D.) riconosce che si tratta dell'istruzione LDA #, ed esegue tutte le abilitazioni necessarie, in particolare:

. PCL (che contiene $00) scarica su DATA BUS interno.

. ALU incrementa DATA BUS ($00) e presenta il risultato ($01) su ADL interno.

. PCL carica da ADL ($01): e' stato incrementato il PROGRAM COUNTER.

. Il BUS indirizzi contiene ora $1001. I.D. attende la transizione di Φ0 per accedere all'indirizzo di memoria specificato.

. Passata la transizione I.D. disabilita l'uscita dell'ALU su

ADL e l'uscita di PCL sul BUS DATI interno, abilita DATA BUS BUFFER a presentare sul BUS DATI interno il contenuto del BUS DATI esterno ($24, il contenuto del byte $1001).

. I.D. abilita l'accumulatore a caricare dal BUS DATI interno ($24). E' stato caricato $24 nell'accumulatore.

. I.D. disabilita l'uscita dal DATA BUS BUFFER, abilita PCL ($01) a scaricare sul BUS DATI interno e l'ALU incrementa il contenuto del BUS DATI interno, presentando il risultato ($02) su ADL.

. PCL viene abilitato a caricare da ADL, e intanto viene effettuato un accesso alla locazione $1002 (si attende una transizione di Φ0) per leggere il codice dell'istruzione successiva.

Non ti spaventare se hai trovato difficile seguire tutte queste operazioni: abbiamo affrontato un argomento abbastanza complicato. Non e' essenziale capire perfettamente come funziona dentro la CPU per usarla, tuttavia pensiamo sia utile avere almeno un'idea di come si svolgono le operazioni all'interno di questa "scatola nera". Puoi soffermarti su questi argomenti in una seconda lettura: ti consigliamo di seguire sulla Figura 8.3 tut-

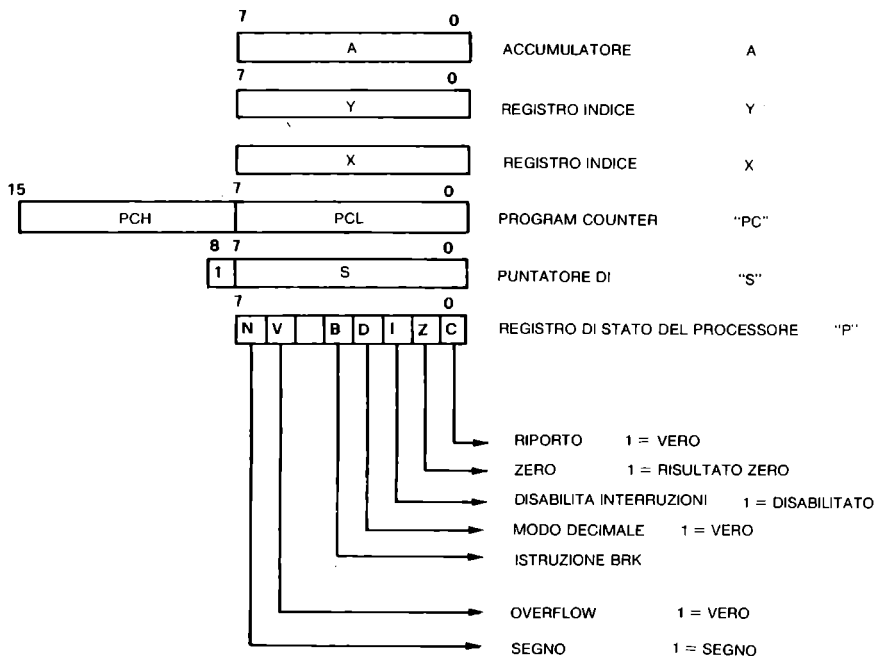



Figura 8.4 Registri della CPU 7501 a disposizione del programmatore

te le operazioni, se vuoi capire bene l'esempio appena spiegato: in seguito puoi anche provare per esercizio (e per divertimento) a far eseguire alla CPU qualche altra istruzione.
Soffermiamo ora l'attenzione sui FLAG. Abbiamo accennato che vi e' un registro a 8 bit chiamato STATO DEL PROCESSORE che contiene 7 FLAG. Un FLAG e' un bit che ha un certo significato logico. FLAG in inglese vuol dire "bandiera": e' come una bandiera che si alza ad indicare che un determinato evento si e' verificato.
Nella Figura 8.4 trovi riassunti i nomi dei registri della 7501. Puoi osservare come sia organizzato il REGISTRO DI STATO:

. BIT 7 = N. Flag di SEGNO. Questo bit e' a 1 se il risultato dell'operazione e' negativo. La 7501 tratta i numeri negativi in COMPLEMENTO A DUE. Secondo tale rappresentazione il bit piu' significativo rappresenta il segno, (0=positivo, 1=negativo). I numeri negativi sono rappresentati come "parte che manca al valore assoluto del numero per arrivare a 256" (256 nei numeri a 8 bit; 65536 nei numeri a 16 bit; 2^n nei numeri a n bit). Prendiamo il numero -12. Quanto manca da 12 per arrivare a 256? Manca 244; 244 si rappresenta in binario puro 11110100. Questa e' anche la rappresentazione di -12 in complemento a 2. Tale rappresentazione a prima vista appare assai scomoda, ma e' in realta' la piu' efficiente dal punto di vista dell'elaborazione. Il FLAG DI SEGNO riporta lo stato del bit piu' significativo del risultato. Il suo valore viene aggiornato dopo le istruzioni di somma, sottrazione, incremento, decremento, caricamento da memoria, confronto, rotazione di bit, scorrimento di bit, trasferimento da registro a registro, ecc. La tabella nell'Appendice A mostra nella sesta colonna da destra quali sono le istruzioni che alterano lo stato di questo FLAG.

. BIT 6 = V. Overflow. Questo bit indica se c'e' stato traboccamento nelle operazioni in complemento a 2, cioe' se il risultato e' troppo grande (>127) o troppo piccolo (<-128) per essere rappresentato in complemento a 2 con 8 bit. In questo caso il risultato e' sbagliato. Nell'aritmetica in valore assoluto e nei byte meno significativi dei numeri in complemento a 2 che occupano piu' di un byte, lo stato di questo bit non ha nessun significato. Solamente somma e sottrazione alterano lo stato di questo FLAG, oltre all'istruzione BIT (in questo caso V assume un significato diverso, e indica lo stato del bit 6 del risultato), e CLV che pone a 0 lo stato di questo bit.

. BIT 5 = non utilizzato

. BIT 4 = B. Istruzione BRK. L'istruzione BRK e' molto particolare: quando viene incontrata la CPU esegue le stesse operazioni di quando riceve un interrupt (vedi Paragrafi 8.5 e 7.3), con la differenza che il registro di stato salvato nello stack contiene 1 nella posizione del bit di BREAK. Normalmente nella programmazione non viene usato. Poiche' non esistono istruzioni che permettono di vedere lo stato di questo FLAG nella CPU, biso-

gna prelevarlo dallo stack con una piccola routine. Puoi vedere questa routine disassemblando il sistema operativo, partendo dall'indirizzo $CE00. Se il FLAG di BRK e' a 1 il SISTEMA OPERATIVO fa un salto indiretto a $0316, se no a $0314.

. BIT 3 = D. Modo decimale. Abbiamo visto nel precedente paragrafo che rappresentando i numeri col modo BCD (decimale a codifica binaria) le operazioni aritmetiche devono essere eseguite con un criterio diverso rispetto al binario puro. Questo FLAG, che e' controllato dal programmatore, informa l'ALU che i conti vanno fatti in BCD. L'istruzione SED attiva il modo BCD, e CLD lo disattiva.

. BIT 2 = I. Disabilita le interruzioni: questo bit e' sotto il controllo diretto del programmatore, cioe' non cambia in base al risultato di operazioni; lo si puo' porre a 1 o a 0, l'istruzione SEI lo pone a 1, l'istruzione CLI lo pone a 0. Come tutti gli altri FLAG il suo valore viene aggiornato anche dopo le istruzioni PLP e RTI, che prelevano lo stato del processore dallo STACK.

. BIT 1 = Z. Risultato zero. Questo e' uno dei FLAG piu' usati nei programmi in ASSEMBLER: indica se tutti gli 8 bit del risultato sono a 0, e solo in tal caso Z vale 1. Come per il FLAG N, anche il FLAG Z viene modificato da numerose istruzioni, come puoi osservare nella quinta colonna da destra dell'Appendice A.

. BIT 0 = C. Carry, o riporto o prestito. Questo FLAG viene alterato dal risultato di diverse operazioni, e talvolta con significati diversi. Nella somma indica se il risultato ha superato 255, e nella sottrazione indica se c'e' stato prestito. Puoi vedere quali operazioni interessano il FLAG C nella quarta colonna da destra della tabella nell'Appendice B.

Abbiamo nominato i termini STACK e STACK POINTER, senza descrivere il loro significato: lo STACK e' un'area di memoria riservata alla CPU dove la CPU pone delle informazioni in maniera temporanea. STACK vuol dire "catasta", e il suo funzionamento e' paragonabile a quello di una catasta o, piu' elegantemente, di una struttura di dati di tipo LIFO (Last In, First Out, cioe' l'ultimo dato entrato e' il primo che esce). Nella 7501 lo STACK occupa le locazioni da $100 a $1FF, cioe' la pagina 1 della memoria. Lo STACK POINTER e' un registro della CPU di 8 bit, che indica la parte meno significativa del primo byte libero dello STACK (la parte piu' significativa e' sempre $01). All'accensione lo STACK POINTER viene posto a $FF. Ogni volta che si spinge un dato nello STACK (PUSH) il dato viene posto all'indirizzo specificato dallo STACK POINTER, dopodiche' lo STACK POINTER viene decrementato.

Nel prelevare un dato dallo STACK (PULL) viene prima incrementato lo STACK POINTER e poi prelevato il dato. Lo STACK e' molto importante per il funzionamento del calcolatore: l'istruzione JSR (salta alla subroutine), ad esempio, usa lo STACK per

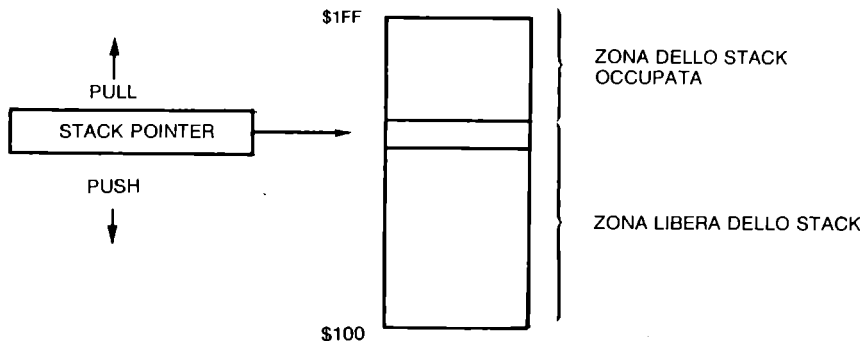



Figura 8.5 STACK e STACK POINTER

memorizzare l'indirizzo dove tornare al momento dell'istruzione RTS, ed anche l'INTERRUPT fa uso dello STACK per poter riprendere il programma alla fine della routine di servizio dell'interrupt. Vi sono diverse istruzioni che fanno uso dello STACK, come PHP, PLP, PHA, PLA, RTS, RTI, BRK; inoltre lo STACK POINTER puo' essere trasferito nel registro X (TSX) e viceversa (TXS). Vedi il Paragrafo 8.4 per la descrizione del funzionamento di queste istruzioni.
Abbiamo gia' visto la funzione dei registri STATO del processore, STACK POINTER, PROGRAM COUNTER. Vediamo ora l'ACCUMULATORE. Come puoi notare dalla Figura 8.3 l'ACCUMULATORE e' collegato direttamente all'ALU. E' questo il registro a cui fa riferimento gran parte delle istruzioni aritmetico-logiche. Sull'accumulatore e' possibile effettuare scorrimento di bit, operazioni logiche AND, OR, EOR (Exclusive OR), somma, sottrazione.
I due registri indice X e Y sono usati come indici negli indirizzamenti indicizzati, come puoi meglio osservare nel prossimo paragrafo.

8.3 I MODI DI INDIRIZZAMENTO

Nella descrizione della 7501 abbiamo detto che tale CPU ha molti modi di indirizzamento, e molto potenti; ora passiamo a descriverli.
Il modo di indirizzamento in una CPU e' il modo con cui e' possibile ricavare l'indirizzo dell'operando. Poiche' ogni istruzione ha lo scopo di produrre qualche effetto su qualche operando (registro della CPU o cella di memoria), possiamo dedurre che ogni istruzione (anche NOP, che fa cambiare il PROGRAM COUNTER per eseguire l'istruzione seguente) usa un modo di indiriz-

zamento. Nella prima riga dell'Appendice A sono riportati tutti i modi di indirizzamento della 7501. VEDIAMOLI UNO PER UNO.

. IMMEDIATO. L'operando segue l'istruzione. Si indica con il segno # prima dell'operando. Ad esempio:

```
LDA #$24
```

si codifica in memoria cosi':

```
$A9 $24
```

Nell'accumulatore viene posto il numero $24.

. ASSOLUTO. L'operando e' contenuto in memoria all'indirizzo specificato dai due byte seguenti il codice dell'istruzione:

```
LDA $134F
```

si codifica in memoria cosi':

```
$AD $4F $13
```

Nell'accumulatore viene posto il numero contenuto nel byte di indirizzo $134F (prima il byte basso e poi quello alto).

. PAGINA ZERO. L'operando e' contenuto in memoria all'indirizzo della pagina 0 specificato dal byte che segue l'istruzione:

```
LDA $24
```

si codifica in memoria cosi':

```
$A5 $24
```

A differenza dell'immediato, nell'accumulatore viene posto il byte il cui indirizzo e' contenuto nel byte di indirizzo $0024 (e non il numero $24).

. ACCUMULATORE. L'operando e' dentro la CPU, e precisamente nell'accumulatore:

```
LSR
```

si codifica in memoria cosi':

```
$4A
```

Viene fatto scorrere verso destra il contenuto dell'accumulatore.

. IMPLICATO. L'operando e' sottinteso dal tipo di istruzione:

```
CLC
```

si codifica in memoria cosi':

```
$18
```

Azzera il CARRY. L'operando e' palesemente il CARRY.

. ASSOLUTO,X. L'operando e' contenuto in memoria, e l'indirizzo si ricava sommando ai due byte, che seguono il codice dell'istruzione, il contenuto del registo X:

```
LDA $103D,X
```

si codifica in memoria cosi':

```
$BD $3D $10
```

Supponiamo che il registro X contenga $D1: $103D e' sommato a $D1, ottenendo $110E. L'operando e' contenuto all'indirizzo $110E.

. ASSOLUTO,Y. Opera come ASSOLUTO,X, ma anziche' il registro X usa il registro Y:

```
LDA $103D,Y
```

si codifica in memoria cosi':
  $B9 $3D $10.
  . PAGINA 0,X L'operando si trova in pagina 0, all'indirizzo ottenuto sommando il byte seguente l'istruzione e il registro X:
  LDA $F0,X
si codifica in memoria cosi':
  $B5 $F0
Se X contiene $32, nell'accumulatore viene posto il byte contenuto nella cella $0022 ($F0 + $32 =$122, ma l'operando viene prelevato sempre dalla pagina 0).
  . PAGINA 0,Y. Opera come PAGINA 0,X, ma usa il registro Y. E' implementato solo con le istruzioni LDX e STX:
  LDX $20,Y
si codifica in memoria cosi':
  $B6 $20.
  . INDIRETTO. L'istruzione e' seguita da due byte che indicano l'indirizzo del puntatore dove si trova l'indirizzo dell'operando:
  JMP ($2000)
si codifica in memoria cosi':
  $6C $00 $20
Produce un salto non alla cella $2000, ma all'indirizzo specificato nelle celle $2000 e $2001. Se tali celle contengono i numeri $12 e $34 rispettivamente, la CPU salta all'indirizzo $3412 (il byte piu' significativo e' sempre dopo quello meno significativo).
  . (INDIRETTO),Y. Si chiama INDIRETTO INDICIZZATO, ed e' uno dei modi piu' potenti. L'istruzione e' seguita da un byte, che indica l'indirizzo in pagina 0 del puntatore al dato. All'indirizzo contenuto nel puntatore va sommato il contenuto del registro Y per ricavare l'indirizzo dell'operando:
  LDA ($03),Y
si codifica in memoria cosi':
  $B1 $03
Se le celle $0003 e $0004 contengono rispettivamente $14 e $2E e il registro Y contiene $F1, l'indirizzo che risulta e' il seguente: $2E14 + $F1 = $2F08. Viene caricato l'accumulatore col contenuto della cella $2F08.
  . (INDIRETTO,X). Si chiama INDICIZZATO INDIRETTO, e differisce dal precedente. Il contenuto del registro X e' sommato al byte seguente l'istruzione, il risultato indica l'indirizzo di un puntatore in pagina 0. L'operando si trova all'indirizzo specificato dal puntatore suddetto:
  LDA ($F0,X)
si codifica in memoria cosi':
  $A1 $F0
Se X contiene $31 viene sommato $F0 con $31, ottenendo $121. E' considerata la parte meno significativa, e l'indirizzo dell'o-

perando e' prelevato dalle celle $21 e $22.

. RELATIVO. Questo indirizzamento e' usato per tutte le istruzioni di salto condizionato, o BRANCH. L'operando, che segue l'istruzione, e' un byte considerato come un numero in complemento a due (puo' variare da -128 a +127). L'operando e' sommato al PROGRAM COUNTER e il risultato ottenuto e' immagazzinato nel PROGRAM COUNTER. L'operando indica lo "spiazzamento" da fornire al PROGRAM COUNTER nel caso in cui si verifica la condizione di salto.

```
BCC $09
```

si codifica in memoria cosi':

```
$90 $09
```

e produce l'effetto seguente: se il CARRY e a 1 prepara il P.C. per eseguire l'istruzione che segue, se invece CARRY = 0 al P.C viene sommato 9 rispetto sempre all'istruzione che segue, e il programma riprende dal nono byte che segue l'istruzione dopo BRANCH (ma non necessariamente dalla nona istruzione, perche' una istruzione puo' occupare uno, due o tre byte). Nota che usando il MONITOR devi fornire l'indirizzo assoluto, e il MONITOR calcola l'indirizzo relativo. Il vantaggio di questo tipo di indirizzamento e' che puoi trasferire il programma in un'altra zona della memoria senza cambiare gli indirizzi dei salti.

Abbiamo analizzato tutti i modi possibili di indirizzamento della CPU 7501. Non occorre impararli tutti a memoria subito, ma ti consigliamo di rileggere quelli che non hai capito prima di avventurarti nella scrittura di programmi in linguaggio ASSEMBLER.

8.4 IL SET DI ISTRUZIONI

Nell'esposizione del completo set di istruzioni del microprocessore 7501, distingueremo cinque categorie principali:

1) .trasferimento dati,
2) .logico matematiche,
3) .controllo del flusso,
4) .manipolazione dei flag,
5) .uso dello stack.

.1): Consentono di trasferire dati di 8 bit da un registro all'altro, da un registro alla memoria o viceversa.

.2): Permettono al microprocessore di eseguire operazioni aritmetiche (piu' e meno), operazioni logiche (AND, OR, EOR (exclusive OR)), operazioni di scorrimento (shift e rotazioni), incremento e decremento.

.3): Sono usate per i salti condizionati, salti non condizionati, salti a subroutine e gestione dell'interrupt.
.4): Consentono di agire sul registro di stato del processore.
.5): Permettono di introdurre ed estrarre dati dallo stack, senza doverti preoccupare della gestione dello stack pointer.
Nel corso dell'esposizione compaiono alcuni simboli di cui riportiamo il significato:

A: accumulatore
M: cella di memoria (byte di memoria)
P: registro di stato
S: stack pointer
X: registro X
Y: registro Y
DATO: dato specificato
PC: program counter
STACK: l'ultima cella dello stack
=: assegnamento
$\wedge$: AND logico
V: OR logico
$\underline{\forall}$: EOR (OR esclusivo)
$\overline{C}$: NOT CARRY
(...): contenuto di ...
BN: bit N
MN: bit N della cella (byte) di memoria specificata.
V: FLAG di overflow
N: FLAG di segno
I: FLAG di interrupt
D: FLAG decimale
C: FLAG di carry
Z: FLAG di zero

8.4.1 LE OPERAZIONI LOGICHE

Prima di illustrare le istruzioni del set della CPU 7501, vorremmo spendere qualche parola riguardo alle operazioni logiche AND OR ed EOR (exclusive OR).
Hai gia' incontrato le prime due in BASIC e le hai usate per fare di piu' condizioni una sola condizione, ad esempio:

```
IF (A AND Z<1) OR A$="PIPPO" THEN ...
```

l'istruzione dopo il THEN viene eseguita se sono vere entrambe le prime due condizioni o se e' vera la terza: cioe' se e' vera la condizione composta.
Ma le operazioni logiche AND e OR sono piu' generali di quanto non possa sembrare dall'esempio appena fatto: esse possono agire, infatti, anche tra numeri interi, oltre che tra condizioni

vere o false.
Cominciamo a definire le operazioni logiche tra bit:

```
0 AND 0 = 0    0 OR 0 = 0    0 EOR 0 = 0
0 AND 1 = 0    0 OR 1 = 1    0 EOR 1 = 1
1 AND 0 = 0    1 OR 0 = 1    1 EOR 0 = 1
1 AND 1 = 1    1 OR 1 = 1    1 EOR 1 = 0
```

Il risultato della AND tra due bit e' quindi uguale a 1 solo se il primo E il secondo bit sono uguali a 1; il risultato della OR tra due bit e' uguale a uno se lo sono il primo O il secondo (o entrambi); il risultato della EOR tra due bit e' uguale a 1 se lo sono il primo O il secondo (ma non entrambi).
Estendere ora il significato dell'operazione a numeri interi di X bit e' semplice: il bit N del risultato di un'operazione logica tra byte e' uguale al risultato dell'operazione logica tra i bit corrispondenti dell'operando; ad esempio:

```
 2 AND 3 = 2
   infatti:
00000010 AND
00000011 =
00000010 cioe' 2
```

Le AND e OR del BASIC sono proprio le operazioni logiche che abbiamo descritto.
Prova a chiedere al tuo COMMODORE PLUS-4:

```
PRINT 2 AND 3
```

e vedrai che il risultato sara' corretto. Queste operazioni vanno bene anche per legare tra di loro condizioni perche' il COMMODORE PLUS-4 assegna ad un'espressione vera il valore -1 (in complemento a due su 16 bit = 16 bit a 1) e ad una falsa il valore 0 (16 bit a 0) (prova a scrivere PRINT 2>7 o PRINT 2<7): quando deve combinare tra loro due condizioni con la funzione AND il risultato e' -1 (cioe' vero) solo se entrambe le condizioni valgono -1 (cioe' sono vere); se l'operazione e' la OR il risultato e' -1 se almeno una delle due vale -1.
A questo punto sorge spontanea una domanda: perche' nel set di istruzioni di un microprocessore appaiono queste strane operazioni mentre non compaiono operazioni piu' usuali come la moltiplicazione e la divisione? Perche' queste operazioni ti permettono di leggere o alterare un singolo bit o un gruppo di bit di un byte, cosa questa assai piu' importante, in molti casi, di una moltiplicazione o una divisione (che si possono comunque ottenere grazie a un opportuno algoritmo). Ad esempio: vuoi sapere quanto vale il bit 3 di un certo byte? Basta fare la AND

tra questo byte e 8 (2^3): se il risultato e' 0 il bit vale 0, se e' 8 il bit e' a 1. Altro esempio: vuoi porre a 1 il bit 5? Fai la OR con 32 (2^5). Se lo volevi a 0? Fai la AND con 223 (255-32 cioe' un numero che ha tutti i bit a uno tranne il quinto).La EOR a cosa serve? Serve a cambiare il valore di uno o piu' bit. Ad esempio 01010011 EOR 00001111 = 01011100: questa operazione ha cambiato il valore dei bit 3-0 del primo operando (cioe' quelli che valgono 1 nel secondo operando). Se fai una EOR tra il contenuto di un byte e $FF "neghi" il contenuto del byte: cioe' i bit che valevano 1 valgono 0 e viceversa.
Ti proponiamo il programma AND&OR che genera operazioni logiche casuali tra numeri binari di 8 bit. Dopo aver calcolato la risposta premi un tasto e il tuo calcolatore ti dara' la conferma.

```
0 REM AND&OR
10 A=INT(RND(0)*256):B=INT(RND(0)*256):OP=INT(RN
D(0)*2)
20 PRINTCHR$(147)
30 FORI=0TO7
40 IFAAND2↑(7-I)THENCHAR,4+I,3,"●":ELSECHAR,4+I,
3,"o"
50 NEXTI:CHAR,12,3,STR$(A)
60 IFOPTHENCHAR,1,4,"OR":ELSECHAR,0,4,"AND"
70 FORI=0TO7
80 IF B AND2↑(7-I)THENCHAR,4+I,4,"●":ELSECHAR,4+
I,4,"o"
90 NEXTI:CHAR,12,4,STR$(B)
100 CHAR,2,6,"="
110 IFOPTHENR=AORB:ELSER=AANDB
115 GETKEYA$
120 FORI=0TO7
130 IFRAND2↑(7-I)THENCHAR,4+I,6,"●":ELSECHAR,4+I
,6,"o"
140 NEXTI:CHAR,12,6,STR$(R)
150 GETKEYA$:RUN
```

8.4.2 ISTRUZIONI DI TRASFERIMENTO

LDA: LoaD Accumulator A=DATO
Il dato specificato viene posto nell'accumulatore.
INDIRIZZAMENTO: immediato - assoluto - pagina 0 - indicizzato X e Y - pagina 0 indicizzato X - indiretto indicizzato - indicizzato indiretto.
FLAG MODIFICATI: zero e segno.

LDX: LoaD X register X=DATO
Il dato specificato viene posto nel registro X.

INDIRIZZAMENTO: immediato - assoluto - pagina 0 - indicizzato Y - pagina 0 indicizzato Y.
FLAG MODIFICATI: zero e segno.

LDY: LoaD Y register Y=DATO
Il dato specificato viene posto nel registro Y.
INDIRIZZAMENTO: immediato - assoluto - pagina 0 - indicizzato X - pagina 0 indicizzato X.
FLAG MODIFICATI: zero e segno.

STA: STore Accumulator M=(A)
Il contenuto dell'accumulatore viene posto nella cella di memoria di indirizzo specificato. Il contenuto dell'accumulatore non viene cambiato.
INDIRIZZAMENTO: assoluto - pagina 0 - indicizzato X e Y - pagina 0 indicizzato X - indiretto indicizzato - indicizzato indiretto.
FLAG MODIFICATI: nessuno.

STX: STore X register M=(X)
Il contenuto del registro X viene posto nella cella di memoria di indirizzo specificato. Il contenuto del registro X non viene cambiato.
INDIRIZZAMENTO: assoluto - pagina 0 - pagina 0 indicizzato Y.
FLAG MODIFICATI: nessuno.

STY: STore Y register M=(Y)
Il contenuto del registro Y viene posto nella cella di memoria di indirizzo specificato. Il contenuto del registro Y non viene cambiato.
INDIRIZZAMENTO: assoluto - pagina 0 - pagina 0 indicizzato X.
FLAG MODIFICATI: nessuno.

TAX: Transfer Accumulator to X register X=(A)
Il contenuto dell'accumulatore viene posto nel registro X. Il contenuto dell'accumulatore non viene cambiato.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: zero e segno.

TAY: Transfer Accumulator to Y register Y=(A)
Il contenuto dell'accumulatore viene posto nel registro Y. Il contenuto dell'accumulatore non viene cambiato.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: zero e segno.

TSX: Transfer Stack pointer to X register X=(S)
Il contenuto dello stack pointer viene posto nel registro X. Il contenuto dello stack pointer non viene cambiato.

INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: zero e segno.

TXA: Transfer X register to Accumulator A=(X)
Il contenuto del registro X viene posto nell'accumulatore. Il contenuto del registro X non viene cambiato.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: zero e segno.

TXS: Transfer X register to Stack pointer S=(X)
Il contenuto del registro X viene posto nello stack pointer. Il contenuto del registro X non viene cambiato.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: nessuno.

TYA: Transfer Y register to Accumulator A=(Y)
Il contenuto del registro Y viene posto nell'accumulatore. Il contenuto del registro Y non viene cambiato.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: zero e segno.

8.4.3 OPERAZIONI LOGICO MATEMATICHE.

ADC: ADd with Carry A=(A)+DATO+C
Somma il contenuto dell'accumulatore con il dato specificato e il carry. Il risultato viene posto nell'accumulatore.
INDIRIZZAMENTO: immediato - assoluto - pagina 0 - indicizzato X e Y - pagina 0 indicizzato X - indiretto indicizzato - indicizzato indiretto.
FLAG MODIFICATI: carry, zero, overflow e segno.

AND: AND A=(A)∧DATO
Esegue l'AND logico tra l'accumulatore e il dato specificato. Il risultato viene posto nell'accumulatore.
INDIRIZZAMENTO: immediato - assoluto - pagina 0 - pagina 0 indicizzato X - indicizzato X e Y - indiretto indicizzato - indicizzato indiretto.
FLAG MODIFICATI: zero e segno.

ASL: Arithmetic Shift Left C=B7=B6=B5=B4=B3=B2=B1=B0=0
Sposta il contenuto dell'accumulatore o della cella di memoria specificata di una posizione bit verso sinistra. Il contenuto del bit 0 diventa 0 e il contenuto del bit 7 viene posto nel flag di carry. Questa operazione equivale ad eseguire una moltiplicazione per 2.
INDIRIZZAMENTO: accumulatore - assoluto - pagina 0 - indicizzato

X - pagina 0 indicizzato X.
FLAG MODIFICATI: carry, zero e segno.

BIT: test BITs in memory (A)∧(M) - N=(M7) - V=(M6)
Esegue la AND logica tra il contenuto dell'accumulatore e il contenuto della cella di memoria specificata. Il risultato non viene posto da nessuna parte ma viene alterato il flag di zero. Pone il contenuto del bit 7 della cella di memoria specificata nel flag di segno e il contenuto del bit 6 nel flag di overflow.
INDIRIZZAMENTO: assoluto - pagina 0.
FLAG MODIFICATI: zero, overflow e segno.

CMP: CoMPare with accumulator (A)-DATO
Sottrae al contenuto dell'accumulatore il dato specificato. Il risultato non viene posto da nessuna parte ma vengono alterati i flag di carry, zero e segno (C=1 indica A>=DATO, Z=1 indica A=DATO).
INDIRIZZAMENTO: immediato - assoluto - pagina 0 - indicizzato X e Y - pagina 0 indicizzato X - indiretto indicizzato - indicizzato indiretto.
FLAG MODIFICATI: carry, zero e segno.

CPX: ComPare with X register (X)-DATO
Sottrae al contenuto del registro X il dato specificato. Il risultato non viene posto da nessuna parte ma vengono alterati i flag di carry, zero e segno (C=1 indica A>=DATO, Z=1 indica A=DATO).
INDIRIZZAMENTO: immediato - assoluto - pagina 0
FLAG MODIFICATI: carry, zero e segno.

CPY: ComPare with Y register (Y)-DATO
Sottrae al contenuto del registro Y il dato specificato. Il risultato non viene posto da nessuna parte ma vengono alterati i flag di carry, zero e segno (C=1 indica A>=DATO, Z=1 indica A=DATO).
INDIRIZZAMENTO: immediato - assoluto - pagina 0
FLAG MODIFICATI: carry, zero e segno.

DEC: DECrement M=(M)-1
Decrementa il contenuto della cella di memoria specificata. Il risultato viene posto nella cella stessa.
INDIRIZZAMENTO: assoluto - pagina 0 - indicizzato X - pagina 0 indicizzato X.
FLAG MODIFICATI: zero e segno.

DEX: DEcrement X register X=(X)-1
Decrementa il contenuto del registro X. Il risultato viene posto

nello stesso registro.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: zero e segno.

DEY: DEcrement Y register Y=(Y)-1
Decrementa il contenuto del registro Y. Il risultato viene posto nello stesso registro.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: zero e segno.

EOR: Exclusive OR A=(A)⊻DATO
Esegue l'OR esclusivo tra l'accumulatore e il dato specificato. Il risultato viene posto nell'accumulatore.
INDIRIZZAMENTO: immediato - assoluto - pagina 0 - pagina 0 indicizzato X - indicizzato X e Y - indiretto indicizzato - indicizzato indiretto.
FLAG MODIFICATI: zero e segno.

INC: INCrement M=(M)+1
Incrementa il contenuto della cella di memoria specificata. Il risultato viene posto nella cella stessa.
INDIRIZZAMENTO: assoluto - pagina 0 - indicizzato X - pagina 0 indicizzato X.
FLAG MODIFICATI: zero e segno.

INX: INcrement X register X=(X)+1
Incrementa il contenuto del registro X. Il risultato viene posto nello stesso registro.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: zero e segno.

INY: INcrement Y register Y=(Y)+1
Incrementa il contenuto del registro Y. Il risultato viene posto nello stesso registro.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: zero e segno.

LSR: Logic Shift Right C=B0=B1=B2=B3=B4=B5=B6=B7=0
Sposta il contenuto dell'accumulatore o della cella di memoria specificata di una posizione bit verso destra. Il contenuto del bit 7 diventa 0 e il contenuto del bit 0 viene posto nel flag di carry. Questa operazione equivale ad eseguire una divisione intera per 2.
INDIRIZZAMENTO: accumulatore - assoluto - pagina 0 - indicizzato X - pagina 0 indicizzato X.
FLAG MODIFICATI: carry, zero e segno.

ORA: OR Accumulator A=(A)VDATO

Esegue l'OR logico tra l'accumulatore e il dato specificato. Il risultato viene posto nell'accumulatore.
INDIRIZZAMENTO: immediato - assoluto - pagina 0 - indicizzato X e Y - pagina 0 indicizzato X - indiretto indicizzato - indicizzato indiretto.
FLAG MODIFICATI: zero e segno.

ROL: ROtate Left C=B7=B6=B5=B4=B3=B2=B1=B0=C
Ruota il contenuto dell'accumulatore o della cella di memoria specificata di una posizione bit verso sinistra. Nel bit 0 viene posto il contenuto del flag di carry, nel quale viene trasferito il contenuto del bit 7.
INDIRIZZAMENTO: accumulatore - assoluto - pagina 0 - indicizzato X - pagina 0 indicizzato X.
FLAG MODIFICATI: carry, zero e segno.

ROR: ROtate Right C=B0=B1=B2=B3=B4=B5=B6=B7=C
Ruota il contenuto dell'accumulatore o della cella di memoria specificata di una posizione bit verso destra. Nel bit 7 viene posto il contenuto del flag di carry, nel quale viene trasferito il contenuto del bit 0.
INDIRIZZAMENTO: accumulatore - assoluto - pagina 0 - indicizzato X - pagina 0 indicizzato X.
FLAG MODIFICATI: carry, zero e segno.

SBC: SuBtract with Carry A=(A)-DATO-$\overline{C}$
Sottrae al contenuto dell'accumulatore il dato specificato. Il risultato viene posto nell'accumulatore. Devi usare il carry al contrario di come lo usi per l'istruzione ADC: carry a 0 vuol dire prestito; devi quindi porre a 1 il carry prima di eseguire un'operazione SBC senza prestito. Dopo un operazione SBC si ha: C=1 indica A>=DATO, Z=1 indica A=DATO.
INDIRIZZAMENTO: immediato - assoluto - pagina 0 - indicizzato X e Y - pagina 0 indicizzato X - indiretto indicizzato - indicizzato indiretto.
FLAG MODIFICATI: carry, zero, overflow e segno.

8.4.4 ISTRUZIONI DI CONTROLLO DEL FLUSSO

BCC: Branch on Carry Clear
Salta all'indirizzo specificato se il flag di carry contiene 0
INDIRIZZAMENTO: relativo.
FLAG MODIFICATI: nessuno.

BCS: Branch on Carry Set
Salta all'indirizzo specificato se il flag di carry contiene 1.
INDIRIZZAMENTO: relativo.

FLAG MODIFICATI: nessuno.

BEQ: Branch on EQual
Salta all'indirizzo specificato se il flag di zero contiene 1 (cioe' se il risultato e' 0).
INDIRIZZAMENTO: relativo.
FLAG MODIFICATI: nessuno.

BMI: Branch on MInus
Salta all'indirizzo specificato se il flag di segno contiene 1 (cioe' se il risultato e' negativo).
INDIRIZZAMENTO: relativo.
FLAG MODIFICATI: nessuno.

BNE: Branch on Not Equal
Salta all'indirizzo specificato se il flag di zero contiene 0 (cioe' se il risultato e' diverso da 0).
INDIRIZZAMENTO: relativo.
FLAG MODIFICATI: nessuno.

BPL: Branch on PLus
Salta all'indirizzo specificato se il flag di segno contiene 0 (cioe' se il risultato e' positivo o uguale a 0).
INDIRIZZAMENTO: relativo.
FLAG MODIFICATI: nessuno.

BRK: BReaK
Salva nello stack (PC)+2 e i flag, salta all'indirizzo indicato da $FFFE $FFFF (come per gli interrupt). Per riconoscere se il salto e' stato generato da un interrupt o dall' istruzione BRK devi guardare il flag di break salvato nello stack. BRK viene usato per trovare errori nel programma (come lo STOP del BASIC).
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: break

BVC: Branch on oVerflow Clear
Salta all'indirizzo specificato se il flag di overflow contiene 0 (cioe' se non si e' verificato overflow).
INDIRIZZAMENTO: relativo.
FLAG MODIFICATI: nessuno.

BVS: Branch on oVerflow Set
Salta all'indirizzo specificato se il flag di overflow contiene 1 (cioe' se si e' verificato overflow).

INDIRIZZAMENTO: relativo.
FLAG MODIFICATI: nessuno.

JMP: JuMP
Salta all'indirizzo specificato.
INDIRIZZAMENTO: assoluto - indiretto.FLAG MODIFICATI: nessuno.

JSR: Jump SubRoutine
Salva nello stack (PC)+2 (l'indirizzo prima dell'istruzione che segue JSR) , salta all'indirizzo specificato.
INDIRIZZAMENTO: assoluto.
FLAG MODIFICATI: nessuno.

NOP: No OPeration
Non fa nulla per due cicli di clock. Si usa per cicli di ritardo.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: nessuno.

RTI: ReTurn from Interrupt
Estrae dallo stack il registro di stato e il program counter che erano stati salvati da una chiamata ad interrupt o da una istruzione BRK.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: tutti.

RTS: ReTurn from Subroutine
Estrae dallo stack il program counter che era stato salvato da una istruzione JSR è lo incrementa di 1.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: nessuno.

8.4.5 ISTRUZIONI SUI FLAG.

CLC: CLear Carry C=0
Viene azzerato il flag di carry. Generalmente si usa prima di un'istruzione ADC.
INDIRIZZAMENTO: implicito
FLAG MODIFICATI: carry.

CLD: CLear Decimal D=0
Viene azzerato il flag BCD. Quando questo flag contiene 0, l'ALU esegue le operazioni aritmetiche in binario.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: decimale.

CLI: CLear Interrupt I=0
Viene azzerato il flag di interrupt. Quando questo flag contiene 0 la CPU risponde alle chiamate d'interrupt.

INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: interrupt.

CLV: CLear oVerflow V=0
Viene azzerato il flag di overflow.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: overflow.

SEC: SEt Carry C=1
Viene posto a uno il flag di carry. Generalmente si usa prima di un'istruzione SBC.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: carry.

SED: SEt Decimal D=1
Viene posto a uno il flag decimale. Quando questo flag contiene 1, l'ALU esegue le operazioni aritmetiche in BCD.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: decimale.

SEI: SEt Interrupt I=1
Viene posto a uno il flag di interrupt. Quando questo flag contiene 1 la CPU non risponde alle chiamate d'interrupt.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: interrupt.

8.4.6 OPERAZIONI SULLO STACK

PHA: PusH Accumulator STACK=(A) ; SP=SP-1
Il contenuto dell'accumulatore viene memorizzato nella cella di indirizzo $100+(SP); lo stack pointer viene decrementato.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: nessuno.

PHP: PusH Processor status STACK=(P); SP=SP-1
Il contenuto del registro di stato viene memorizzato nella cella di indirizzo $100+(SP); lo stack pointer viene decrementato.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: nessuno.

PLA: PuLl Accumulator A=(STACK); SP=SP+1
Il contenuto della cella di indirizzo $100+(SP) viene caricato nell'accumulatore; lo stack pointer viene incrementato.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: nessuno.

PLP: PuLl Processor status P=(STACK); SP=SP+1
Il contenuto della cella di indirizzo $100+(SP) viene caricato nel registo di stato; lo stack pointer viene incrementato.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: tutti.

8.5 LA GESTIONE DELL'INTERRUPT

Abbiamo visto nel Paragrafo 7.3 che la CPU possiede un piedino da cui riceve le chiamate di INTERRUPT. Puoi immaginare un segnale di interrupt come la chiamata HARDWARE di un sottoprogramma; la subroutine cioe' non viene eseguita in risposta a una chiamata da programma, ma a un segnale logico che giunge a un piedino della CPU. Il TED e' l'unico dispositivo collegato alla linea di richiesta delle interruzioni; ogni 50-esimo di secondo la CPU riceve da TED una richiesta di interruzione. Il SISTEMA OPERATIVO usa questo segnale per eseguire alcune operazioni a intervalli di tempo regolari, come aggiornare TI$, o memorizzare eventuali caratteri premuti sulla tastiera.
Il programma che segue, in ASSEMBLER, ti mostra come la routine di interrupt, che puo' essere disabilitata, controlla lo stato della tastiera. Puoi introdurlo in memoria usando il MONITOR.

```
A 3000 CLI
A 3001 LDA $C6
A 3003 STA $0C00
A 3006 CLC
A 3007 BCC $3001
```

Per avviarne l'esecuzione puoi scrivere:

```
G 3000.
```

Vedrai che il primo carattere in alto a sinistra nello schermo cambia in funzione del tasto che premi sulla tastiera; questo fatto parrebbe strano sapendo che la cella di indirizzo $C6 e' una normale cella di memoria in pagina 0, e non un registro di ingresso. Se arresti l'esecuzione del programma (l'unico modo e' il tasto di RESET, eventualmente STOP-RESET) e cambi l'istruzione in $3000, cioe' cambi CLC in SEI (mascheri le interruzioni), ti accorgi, avviando nuovamente il programma, che la tastiera non influisce piu' sul quadratino del video. Cio' e' dovuto al fatto che la ROUTINE DI INTERRUPT non viene eseguita quando il FLAG di INTERRUPT e' a 1.
Nel Paragrafo 10.6 riportiamo un sottoprogramma, in linguaggio macchina, che intercetta la routine di servizio dell'INTERRUPT del SISTEMA OPERATIVO.
Quando la CPU risponde a una chiamata di interrupt salva automaticamente nello STACK i seguenti dati:

- .gli 8 bit piu' significativi del P.C.
- .gli 8 bit meno significativi del P.C.
- .il registro di STATO del processore

e produce un salto indiretto a $FFFE.
E' cura del SISTEMA OPERATIVO effettuare tutte le operazioni che si devono eseguire in risposta a un interrupt. Per tornare dalla routine di gestione dell' interrupt esiste l'istruzione RTI, che e' diversa da RTS perche' recupera dallo STACK anche lo stato del processore, oltre a non incrementare il P.C.

8.6 USO DEL COMANDO MONITOR DEL BASIC

La funzione di questo comando BASIC e' quella di portare in ambiente MONITOR il calcolatore. Il MONITOR e' un programma in linguaggio macchina che permette di scrivere facilmente programmi in ASSEMBLER e in linguaggio macchina. Esso comprende un MONITOR di linguaggio macchina, un MINI-ASSEMBLATORE e un DISASSEMBLATORE. I programmi in linguaggio macchina, scritti utilizzando il MONITOR, possono essere utilizzati autonomamente oppure come sottoprogrammi molto veloci per programmi BASIC.

I comandi disponibili da MONITOR sono:
. A ASSEMBLA: assembla un'istruzione all'indirizzo specificato.
. C CONFRONTA: confronta due blocchi della memoria e segnala le differenze.
. D DISASSEMBLA: disassembla il codice della 7501 a partire dall'indirizzo specificato.
. F RIEMPI: riempie la memoria con il numero specificato.
. G ESEGUI: avvia l'esecuzione del programma dall'indirizzo specificato.
. H CERCA: ricerca nella memoria tutte le posizioni di determinati valori di un gruppo di numeri.
. L CARICA: carica un file dal nastro o dal disco.
. M VISUALIZZA MEMORIA: visualizza i valori esadecimali delle locazioni di memoria
R VISUALIZZA REGISTRI: visualizza i registri della 7501, che tu puoi cambiare.
. S SALVA: salva su nastro o su disco un blocco di memoria.
. T TRASFERISCI: trasferisce un blocco di memoria in un'altra posizione.
. V VERIFICA: confronta un blocco della memoria con un file su nastro o su disco.
. X ESCI: esce dal MONITOR e ritorna al BASIC.
. . PUNTO: assembla una riga del codice della 7501.
. > (maggiore di): modifica la memoria.
. ; (punto e virgola): modifica i registri della CPU.

La locazione $7F8 controlla se il MONITOR vede la ROM o la RAM sopra $8000. Se questa locazione contiene 0, il MONITOR visua-

lizza l'interprete BASIC e il KERNAL (routine del SISTEMA OPERATIVO) dopo un comando di disassemblaggio o di stampa, cioe' il contenuto della memoria ROM di indirizzo sopra $8000. Se questa locazione contiene $80, il MONITOR visualizza la RAM che sta sotto l'interprete BASIC e il KERNAL. Cio' non e' particolarmente utile nella macchina originale, ma diventa conveniente per lo sviluppo di programmi in linguaggio macchina per chi possiede l'espansione di memoria da 64K. Nota che la locazione $7F8 non ha alcuna influenza sul comando G. Il comando G avvia l'esecuzione nella mappa di memoria ROM senza tener conto dell'impostazione della locazione $7F8.

Si accede al MONITOR scrivendo: MONITOR
la risposta del sistema e' la visualizzazione dei registri della 7501 e il cursore lampeggiante. Il cursore rappresenta il "prompt" che ricorda che il MONITOR e' in attesa dei comandi.

I comandi, che passiamo a descrivere, fanno uso di parametri; tutti i parametri sono da intendere in forma esadecimale, li scriviamo non preceduti dal dollaro, tra un parametro e l'altro si puo' porre uno spazio oppure una virgola.

A introduce una riga di codice ASSEMBLER.
SINTASSI: A <indirizzo> <codice operativo mnemonico> <operando>
<indirizzo>: locazione della memoria dove collocare il codice operativo.
<codice operativo mnemonico>: codice mnemonico dell'istruzione da assemblare, cosi' come e' descritto nel Paragrafo 8.4 e nell'APPENDICE B.
<operando>: operando, quando richiesto, puo' essere di una qualsiasi delle modalita' d'indirizzamento ammesse. Per le modalita' di indirizzamento in pagina zero si deve indicare un numero esadecimale di 2 cifre; ad esempio: LDA $10. Per indirizzi di pagina-non zero vengono richiesti numeri esadecimali di 4 cifre; ad esempio: LDA $1000. Per l'indirizzamento immediato si deve porre il segno # prima del numero a 2 cifre esadecimale che rappresenta l'operando; ad esempio: LDA #$FF.
Alla fine della riga devi premere RETURN. Se nella riga risultano degli errori, viene visualizzato un punto di domanda ad indicare un errore e il cursore si sposta alla riga seguente. Per correggere eventuali errori puoi tornare su con il cursore e modificare la linea in modo da eliminare l'errore.
Qui di seguito listiamo il programma LEGGIJOYASS, che puoi introdurre in memoria ricopiandolo con il MONITOR, oppure che si carica automaticamente in memoria usando il programma BASIC TEST LEGGEJOYASS.

```
MONITOR
    PC   SR AC XR YR SP
; 0000 00 00 00 00 F8
. 3000  A2 FB     LDX #$FB
. 3002  78        SEI
. 3003  8E 08 FF  STX $FF08
. 3006  AD 08 FF  LDA $FF08
. 3009  8E 08 FF  STX $FF08
. 300C  CD 08 FF  CMP $FF08
. 300F  D0 F2     BNE $3003
. 3011  58        CLI
. 3012  29 4F     AND #$4F
. 3014  C9 10     CMP #$10
. 3016  90 02     BCC $301A
. 3018  09 80     ORA #$80
. 301A  29 8F     AND #$8F
. 301C  49 8F     EOR #$8F
. 301E  AA        TAX
. 301F  A0 FD     LDY #$FD
. 3021  78        SEI
. 3022  8C 08 FF  STY $FF08
. 3025  AD 08 FF  LDA $FF08
. 3028  8C 08 FF  STY $FF08
. 302B  CD 08 FF  CMP $FF08
. 302E  D0 F2     BNE $3022
. 3030  58        CLI
. 3031  29 8F     AND #$8F
. 3033  49 8F     EOR #$8F
. 3035  A8        TAY
. 3036  60        RTS
```

Questa routine ti permette di leggere lo stato dei joystick, per usarli nei tuoi programmi in linguaggio macchina. Come puoi osservare il programma fa uso di un registro di I/O, in particolare il registro $FF08, il cui funzionamento e' stato descritto nel Paragrafo 4.6. Puoi osservare nelle Figura 4.7a e 4.7b come sono collegati i due joystick.
Le istruzioni da $3000 a $3011 caricano nell'accumulatore lo stato dei 5 bit collegati al Joystick 1. Da $3012 a $301E diamo un formato piu' adatto al risultato e lo poniamo nel registro X. Da $301F a $3030 poniamo in A lo stato del joystick 2 e da $3031 a $3036 adattiamo il formato del risultato ponendolo nel registro Y. Alla fine della routine il registro X contiene lo stato del joystick 1, e Y quello del joystick 2.
Nella Figura 8.6 puoi vedere il significato dei bit dei registri X e Y in relazione alle posizioni dei joystick. I bit a 1 indicano che il tasto e' premuto, o la leva e' spostata. Il programma TEST LEGGIJOYSTICK visualizza lo stato dei registri X e Y alla fine della routine. Il codice del programma inoltre e' contenuto nelle linee DATA, e viene posto in memoria nella linea 20.

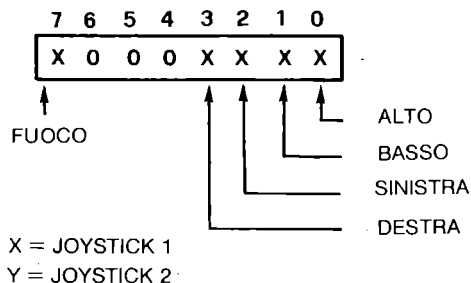



Figura 8.6 Formato dei risultati del sottoprogramma LEGGIJOYASS

E' assai importante non sbagliare neanche uno dei dati, poiche' in tal caso il sistema potrebbe arrestarsi, e non riprendere l'esecuzione neanche premendo RESET. Per questo la linea 25 arresta il programma se la somma dei dati contenuti nelle linee DATA differisce dal valore corretto: ricontrolla bene tutti i dati in caso di errore, e non eliminare la linea 28.

```
0 REM TEST ROUTINE LETTURA JOYSTICK IN ASSEMBLER
5 REM LA ROUTINE VA DA 3000H A 3036H COMPRESI
10 POKE56,48:POKE55,0:CLR
20 FORI=12288TO12342:READA:Q=Q+A:POKEI,A:NEXT
25 IFQ<>7420THENSTOP
30 DO
40 SYS12288:A=PEEK(2035):B=PEEK(2036)
50 GOSUB80:PRINT,
60 A=B:GOSUB80:PRINT
70 LOOP
80 FORI=7TO0STEP-1:IFAAND2↑ITHENPRINT"1";:ELSEPR
INT"0";
90 NEXT:RETURN
100 DATA162,251,120,142,8,255,173,8,255,142,8
105 DATA255,205,8,255,208,242,88,41,79,201
110 DATA16,144,2,9,128,41,143,73,143,170,160
115 DATA253,120,140,8,255,173,8,255,140,8,255
120 DATA205,8,255,208
130 DATA242,88,41,143,73,143,168,96
```

C confronta due blocchi della memoria e segnala le differenze.
SINTASSI: C <indirizzo inizio blocco 1> <indirizzo fine blocco 1> <indirizzo inizio blocco 2>

Esempio: C 1000 1035 2010
Confronta i contenuti delle celle che vanno da $1000 a $1035 compreso, con quelli delle celle che vanno da $2010 a $2045 compreso. Se qualche byte differisce, ne viene stampato l'indirizzo.

D disassembla il contenuto della cella specificata.
SINTASSI: D <indirizzo iniziale> <indirizzo finale>
<indirizzo iniziale>: numero esadecimale di al massimo 4 cifre che indica l'indirizzo dell'istruzione da disassemblare. Se non fornito vengono disassemblati 20 (decimale) byte a partire dall'ultimo disassemblato.
<indirizzo finale>: 4 cifre esadecimali che indicano a quale indirizzo smettere di disassemblare. Se non fornito vengono disassemblati 20 byte.

F riempie una zona di memoria con il numero specificato.
SINTASSI: F <indirizzo iniziale> <indirizzo finale> <dato>
Esempio: F 0C00 0D00 01
riempie la zona di memoria da $0C00 a $0D00 con il numero $01.
Tutti i parametri sono obbligatori.

G avvia l'esecuzione di un programma contenuto in memoria.
SINTASSI: G <indirizzo>
Il parametro <indirizzo> puo' essere omesso: in tal caso il programma parte dall'indirizzo indicato nel registro PC (vedi il comando R).

H ricerca nella memoria le posizioni di determinati byte.
SINTASSI: H <ind. iniz.> <ind. finale> <sequenza di byte> oppure <'sequenza di caratteri>
Cerca nelle locazioni specificate la sequenza di byte specificata.
Esempio: H 1000 2000 1 DE F
cerca se tra $1000 e $2000 e' contenuta la sequenza: $01 $DE $0F.
Se tale sequenza e' presente, viene stampato l'indirizzo del primo byte della sequenza, e se e' presente piu' volte, sono stampati tutti gli indirizzi dove inizia la sequenza specificata.
Esempio: H 1000 4000 'COMMODORE
cerca se tra le celle $1000 e $4000 se e' presente la sequenza di byte il cui codice ASCII compone la stringa "COMMODORE". Nota che i caratteri sono preceduti da un apice (SHIFT-7).

L carica un file da nastro o da disco.
SINTASSI: L <"nomefile"> <numero unita'>

Se non si specifica il numero di unita' si considera il registratore a cassetta. Il file viene caricato al suo indirizzo originale (nella stessa posizione in cui era quando e' stato salvato).
Esempio: L
carica da nastro il primo programma che trova.
Esempio: L "ROUTINE",8
carica in memoria da disco il file chiamato ROUTINE.

M visualizza i valori esadecimali delle locazioni di memoria specificate.
SINTASSI: M <ind. iniz.> <ind. finale>
Gli operandi hanno lo stesso significato che nel comando D. I byte stampati possono essere modificati passandoci sopra col cursore, cambiandoli e premendo RETURN.
Esempio: M 1000
mostra il contenuto di 96 byte a partire dall'indirizzo 1000.

R visualizza il valore dei registri della CPU.
SINTASSI: R
Naturalmente la CPU continua ad eseguire il programma MONITOR e i suoi registri cambiano continuamente. I registri che ti vengono mostrati (e di cui puoi cambiare il contenuto passandoci sopra e battendo RETURN), sono quelli che saranno caricati al momento dell'avvio del programma, e che sono stati prelevati dalla CPU al momento del BRK. L'istruzione BRK porta al MONITOR, e vengono automaticamente mostrati i contenuti dei registri.

S salva su nastro o disco un blocco di memoria.
SINTASSI: S "NOMEPROGRAMMA" <unita'> <indirizzo iniziale> <indirizzo finale>
Tutti i parametri sono obbligatori, tranne il nome del file su cassetta. L'indirizzo finale deve essere uno piu' dell'ultimo byte da salvare:
S "FILE" 08 1001 11F3
Salva su disco il programma di nome FILE a partire dall'indirizzo $1001 fino all'indirizzo $11F2 compreso.

T trasferisce segmenti di memoria da una zona a un'altra.
SINTASSI: T <indir. iniz.> <indir. finale> <nuovo indirizzo>
I dati possono essere trasferiti ovunque nella RAM. Purtroppo un errore del sistema non permette di trasferire un segmento di memoria sopra se stesso.
Esempio: T 1400 1600 1401
non produce l'effetto di spostare di una posizione il segmento verso l'alto nella memoria, ma riempie tutta l'area $1401 $1601 con il contenuto della cella $1400, distruggendo il vecchio

contenuto. Cio' e' dovuto probabilmente all'errore di eseguire il trasferimento del segmento sempre dal basso verso l'alto, mentre sarebbe corretto trasferire dal basso verso l'alto nei trasferimenti dall'alto verso il basso, e viceversa. Prova per esercizio a scrivere un programma che trasferisce blocchi di memoria, e vedrai meglio quali difficolta' incontra il programmatore. La versione del SISTEMA OPERATIVO a cui ci riferiamo e' la numero 4 per la versione PAL (la cella $FF80 della ROM contiene $84). E' possibile che in versioni successive l'errore sia stato eliminato.

V verifica che un file contenga gli stessi dati contenuti in memoria.
SINTASSI: V <"nomefile"> <unita'>
Le regole sono le stesse di L (carica) ma il file non viene caricato in memoria. Se il file differisce dal contenuto della memoria viene stampato il messaggio "VERIFYING ERROR", altrimenti non viene stampato niente.

X torna in ambiente BASIC.
SINTASSI: X

Il MONITOR ti pone in grado di alterare celle di memoria e di compiere molte operazioni pericolose. E' importante non alterare il contenuto di celle di memoria di cui non conosci la funzione, soprattutto nella zona di lavoro dell'interprete BASIC e del SISTEMA OPERATIVO ($0000 - $0800) e nell'I/O ($FD00 - $FFFF). Puo' capitare assai facilmente che il sistema impazzisca a causa di operazioni scorrette sulla memoria. Talvolta il comando X non ti porta al BASIC: quando cio' capita e' perche' il sistema e' in umo stato anomalo, da cui puo' uscire solo con un RESET.

CAPITOLO 9

LA GRAFICA

9.1 INTRODUZIONE

In questo capitolo ti mostriamo alcune caratteristiche avanzate del TED. Grazie alle tecniche esposte ti sara' possibile ottenere nuovi tipi di effetti grafici e di animazione.

9.2 CARATTERI PROGRAMMABILI

Sai gia' come il COMMODORE PLUS-4 possa visualizzare i caratteri sul video, e sai anche che le immagini di questi caratteri (una serie di 8 byte per ogni carattere) sono custodite in ROM perche' non vengano perse ogni volta che spegni il calcolatore. Esiste pero' un modo per "dire" a TED di leggere le descrizioni dei caratteri in RAM e per indicargli l'indirizzo del primo byte del primo carattere. Per far leggere TED dalla RAM devi porre a 0 il bit 2 del registro 12 (indirizzo 65298 ($FF12)) con l'istruzione:

```
POKE 65298, PEEK(65298) AND 251
```

Per indicare l'indirizzo del primo byte del primo carattere, devi porre nei bit 7-2 del registro 13 (indirizzo 65299 ($FF13)) la parte piu' significativa dell'indirizzo stesso, con l'istruzione

```
POKE 65299, (PEEK(65299) AND 3)+N
dove N*256 = indirizzo desiderato
e N e' un numero divisibile per 4
```

Proviamo a programmare un carattere: per esempio il simbolo CBM che e' disegnato sul tasto in basso a sinistra della tastiera. Usando la stessa tecnica con cui sono definiti normalmente i caratteri, costruiamo una matrice 8 per 8 e riempiamola usando 0 per i punti spenti e 1 per quelli accesi:

```
0 0 0 1 1 0 0 0          * *
0 0 1 1 1 0 0 0        * * *
0 1 1 0 0 1 1 1      * *     * * *
0 1 1 0 0 1 1 0      * *     * *
0 1 1 0 0 1 1 1      * *     * * *
0 0 1 1 1 0 0 0        * * *
0 0 0 1 1 0 0 0          * *
0 0 0 0 0 0 0 0
```

Ora dobbiamo fare un po' di conti: la prima riga (cioe' il primo byte del carattere) contiene il numero binario 00011000, cioe' 24 decimale. Facendo lo stesso conto per le altre righe otteniamo i numeri decimali 24, 56, 103, 102, 103, 56, 24, 0. Questi numeri devono essere ora memorizzati nei primi 8 byte della nuova descrizione dei caratteri, programmando cosi' il carattere che ha D/CODE 0, cioe' la chiocciola (vedi Appendice D). Il programma GRAF1 sposta la descrizione dei caratteri all'indirizzo 14336 ($3800), programma il carattere di D/CODE 0 con il simbolo CBM e il carattere di D/CODE 32 (lo spazio) con uno spazio:

```
0 REM GRAF1
10 COLOR0,1:COLOR1,7,5:COLOR4,1
20 POKE65299,(PEEK(65299)AND3)+240
30 POKE65298,PEEK(65298)AND251
40 FORI=0TO7
50 READA:POKE61440+I,A:POKE61696+I,0
60 NEXTI
70 PRINTCHR$(147)CHR$(142)"@"
80 GETKEYA$
90 POKE65299,(PEEK(65299)AND3)+208
100 POKE65298,PEEK(65298)OR4
110 DATA24,56,103,102,103,56,24,0
```

Fai girare il programma: vedrai comparire sullo schermo, in alto a sinistra, il carattere che abbiamo programmato. Se premi un qualunque tasto (tranne STOP) il carattere tornera' ad essere una chiocciola e il programma finira'; se premi il tasto STOP il programma finisce senza riportare la descrizione dei caratteri in ROM: quindi invece di vedere i caratteri normali vedrai, premendo i tasti, dei caratteri casuali (tutti i caratteri di questo nuovo set che non abbiamo ancora programmato).

COMMENTO A GRAF1:
.10: schermo e bordo neri, caratteri azzurri.
.20: descrizione dei caratteri in 14336 ($3800).
.30: TED legge da RAM.
.40/60: programma il carattere di D/CODE 0 con i dati relativi

al simbolo CBM, e il carattere di D/CODE 32 (indirizzo di partenza 14336+32*8 = 14592) con una serie di zeri per ottenere uno spazio.

.70: pulisce lo schermo, seleziona il set di caratteri maiuscolo, e visualizza il carattere di D/CODE 0.

.80: attende che sia premuto un tasto.

.90: descrizione dei caratteri in $D000.

.100: TED legge da ROM.

.110: dati relativi al simbolo CBM.

ATTENZIONE a non fare errori quando TED legge da RAM: infatti quando il sistema da' una segnalazione di errore mette automaticamente TED in condizioni di leggere la ROM e, se l'indirizzo della descrizione dei caratteri era stato cambiato, TED si trovera' a leggere dei dati da indirizzi di ROM non esistenti; il risultato sara' ovviamente spiacevole.

9.3 COPIA DEI CARATTERI DA ROM

Non sempre, quando definisci un nuovo set di caratteri, vuoi ottenere un risultato completamente diverso dal contenuto della ROM: potrebbe capitarti di voler avere a disposizione tutte le lettere del set del COMMODORE PLUS-4 e cambiare solo i caratteri grafici, o cambiare le lettere e conservare le cifre e i segni di punteggiatura. In questo caso dovrai copiare su RAM la parte del contenuto della ROM che ti interessa e poi definire, come hai gia' visto, gli altri caratteri. Purtroppo il BASIC del COMMODORE PLUS-4 quando legge un dato dalla memoria lo legge sempre dalla RAM. L'unico modo di leggere i dati dalla ROM dei caratteri e' quello di usare una semplice routine in linguaggio macchina. Il programma GRAF2 ha il compito di caricare in memoria (dall'indirizzo 8192 ($2000)) il programma GRAF3 e di farlo eseguire:

```
0 REM GRAF2
10 COLOR0,1:COLOR1,7,5:COLOR4,1
20 POKE56,240:POKE55,0:CLR:PRINTCHR$(142)
30 POKE65299,(PEEK(65299)AND3)+240
40 POKE65298,PEEK(65298)AND251
50 FORI=0TO31:READA:POKE8192+I,A:NEXT
60 SYS8192:NEW
1000 DATA169,208,133,4,169,240,133,6
1010 DATA160,0,132,3,132,5,177,3
1020 DATA145,5,200,208,249,230,4,230
1030 DATA6,165,4,201,216,208,239,96
```

```
MONITOR
    PC   SR AC XR YR SP
; 0000  00 00 00 00 F8
. 2000   A9 D0     LDA #$D0
. 2002   85 04     STA $04
. 2004   A9 F0     LDA #$F0
. 2006   85 06     STA $06
. 2008   A0 00     LDY #$00
. 200A   84 03     STY $03
. 200C   84 05     STY $05
. 200E   B1 03     LDA ($03),Y
. 2010   91 05     STA ($05),Y
. 2012   C8        INY
. 2013   D0 F9     BNE $200E
. 2015   E6 04     INC $04
. 2017   E6 06     INC $06
. 2019   A5 04     LDA $04
. 201B   C9 D8     CMP #$D8
. 201D   D0 EF     BNE $200E
. 201F   60        RTS
```

Potrebbe sembrare che il programma GRAF2 non produca alcun effetto; in realta' i dati relativi ai caratteri si trovano ora in RAM. Prova infatti a scrivere una chiocciola sullo schermo e a dare quindi le istruzioni
FOR I=0 TO 7: POKE 14336+I,0: NEXT I.
Appena premi RETURN la chiocciola sparisce perche' hai cancellato in RAM i dati relativi alla chiocciola del set maiuscolo: se selezioni il set minuscolo le chiocciole riappariranno la' dove erano sparite.

COMMENTO A GRAF2
.10: schermo e bordo neri, caratteri azzurri.
.20: aggiorna i puntatori di fine memoria a 14336 ($3800) in modo che le variabili non cancellino i caratteri programmati. Seleziona il set di caratteri maiuscoli.
.30: descrizione dei caratteri in 14336 ($3800).
.40: TED legge da RAM.
.50: pone il programma GRAF3 in memoria
.60: esegue la routine GRAF3 e cancella il programma.
.1000/1030: contengono i dati relativi a GRAF3.

COMMENTO A GRAF3
(con riferimento agli indirizzi esadecimali)
.2000/2002 pone 208 ($D0) nel byte piu' significativo del puntatore 03-04.
.2004/2006 pone 56 ($38) nel byte piu' significativo del puntatore 05-09.
.2008/200C pone 0 ($00) nei byte meno significativi dei due puntatori: 03-04 punta cosi' all'inizio della ROM dei caratteri e

05-06 punta all'inizio dell'area RAM in cui vogliamo trasferire i caratteri.
.200E/2013 trasferisce una pagina ($100 corrisponde a 256 byte) dal byte puntato da 03-04 al byte puntato da 05-09.
.2015/2017 incrementa i piu' significativi dei due puntatori in modo che puntino alla prossima pagina.
.2019/201D se non ha ancora copiato 8 pagine (da $D000 a $D7FF) continua.
.201F torna al BASIC.

Il prossimo programma, GRAF4, ha il compito di trasferire in RAM i caratteri e sostituire le lettere del set MAIUSCOLO/GRAFICO con delle lettere gotiche. Esso usa ancora la routine GRAF3.

```
0 REM GRAF4
10 COLOR0,2,7:COLOR1,1:COLOR4,7,6
20 PRINTCHR$(142)
30 POKE65299,(PEEK(65299)AND3)+240
40 POKE65298,PEEK(65298)AND251
50 FORI=0TO31:READA:POKE8192+I,A:NEXT
60 SYS8192
70 FORI=0TO215:READA:POKE61440+I,A:NEXT
80 SYS32768
1000 DATA169,208,133,4,169,240,133,6
1010 DATA160,0,132,3,132,5,177,3
1020 DATA145,5,200,208,249,230,4,230
1030 DATA6,165,4,201,216,208,239,96
1040 DATA60,102,110,110,96,98,60,0
1050 DATA48,72,20,34,62,34,65,0
1060 DATA92,34,66,124,66,34,92,0
1070 DATA28,34,84,80,80,34,28,0
1080 DATA88,100,66,66,66,100,88,0
1090 DATA92,34,64,112,64,34,92,0
1100 DATA92,34,32,120,32,32,64,0
1110 DATA28,34,64,94,98,62,2,6
1120 DATA28,34,32,60,34,34,36,0
1130 DATA2,60,72,8,10,60,64,0
1140 DATA1,2,2,2,34,68,56,0
1150 DATA66,36,40,112,40,36,66,0
1160 DATA24,36,32,32,32,33,94,0
1170 DATA84,42,42,106,42,42,64,0
1180 DATA66,50,42,106,42,42,68,0
1190 DATA28,34,81,81,81,34,28,0
1200 DATA92,34,34,124,32,32,64,0
1210 DATA56,84,34,2,12,26,124,0
1220 DATA92,34,34,120,36,34,66,0
1230 DATA2,60,64,60,2,60,64,0
1240 DATA1,126,48,80,80,33,30,0
1250 DATA33,82,18,18,18,18,12,0
1260 DATA76,178,34,34,34,20,8,0
1270 DATA128,92,82,82,82,84,40,0
```

```
1280 DATA34,84,12,8,24,37,66,0
1290 DATA66,164,36,36,26,66,60,0
1300 DATA62,2,4,8,16,32,62,0
```

COMMENTO A GRAF4
.10 ripristina le condizioni iniziali dello schermo.
.20 seleziona il set MAIUSCOLO/GRAFICO.
.30/40 indica a TED la nuova descrizione dei caratteri.
.50 carica in memoria GRAF3.
.60 esegue GRAF3.
.70 pone nella memoria dei caratteri i dati riguardanti i caratteri gotici.
.80 inizializza il BASIC.
.1000/1030 dati relativi a GRAF3.
.1040/1300 dati relativi ai nuovi caratteri.

Dopo aver fatto girare questo programma il COMMODORE PLUS-4 e' pronto a funzionare ma puo' rovinare la descrizione dei caratteri con le variabili stringa. Per evitare cio' puoi dare le istruzioni
POKE 56,56: POKE 55,0: CLR
Queste istruzioni abbassano la fine della memoria a 14336 ($3800), dove inizia la descrizione dei caratteri.
Ricorda sempre che un errore in queste condizioni porta TED a leggere da una ROM inesistente. Se ti dovesse capitare di avere una segnalazione di errore e non vuoi resettare puoi dare l'istruzione
POKE 65298,PEEK(65298)AND251
anche se non vedi i caratteri che stai scrivendo; essa resetta il bit di posizione 2 al valore 0.

9.4 PROGRAMMI PER LA CREAZIONE DEI CARATTERI

Programmare nuovi caratteri con il COMMODORE PLUS-4 e' facile ma non molto immediato: e' difficile cioe' "vedere" che 24, 56, 103, 102, 103, 56, 24, 0 rappresentano un simbolo grafico. Sarebbe molto piu' comodo se si potesse disegnare direttamente il carattere sullo schermo e farlo memorizzare senza tanti calcoli, PEEK e POKE. Il programma GRAF5 fa precisamente questo. E' commentato come al solito e puoi quindi eliminare facilmente certe parti che non ti interessano o aggiungerne altre per renderlo piu' adatto ai tuoi scopi.

```
0 REM GRAF5
10 COLOR0,1:COLOR1,7,5:COLOR4,1
15 PRINTCHR$(142)CHR$(8)
```

```
20 POKE56,240:POKE55,0:CLR
30 US$=CHR$(147)+"          PREMI F1 PER USCIRE"
40 KEY1,CHR$(133)
100 PRINTCHR$(147)
110 PRINT"OPZIONI :":PRINT:PRINT
120 PRINT"1 CREA CARATTERE":PRINT
130 PRINT"2 MEMORIZZA CARATTERE":PRINT
140 PRINT"3 CORREGGE CARATTERE":PRINT
150 PRINT"4 MOSTRA CARATTERI":PRINT
160 PRINT"5 SALVA SET DI CARATTERI":PRINT
170 PRINT"6 CARICA SET DI CARATTERI":PRINT
180 PRINT"7 FINE"
190 GETA$:IFVAL(A$)=0THEN190
195 I=VAL(A$)
200 ONIGOSUB1000,2000,3000,4000,5000,6000,7000
210 GOTO100
1000 PRINTUS$
1010 FORI=9TO16:CHAR1,16,I,"++++++++":NEXT
1060 PC=2424:XC=0:YC=0
1070 POKEPC,PEEK(PC)+128
1080 GETKEYA$
1090 A=ASC(A$):SP=0
1100 IFA=29ANDXC<7THENXC=XC+1:PC=PC+1:SP=1
1110 IFA=157ANDXC>0THENXC=XC-1:PC=PC-1:SP=-1
1120 IFA=17ANDYC<7THENYC=YC+1:PC=PC+40:SP=40
1130 IFA=145ANDYC>0THENYC=YC-1:PC=PC-40:SP=-40
1140 IFA=32THENPOKEPC+1024,81
1150 IFA=20THENPOKEPC+1024,43
1160 POKEPC-SP,PEEK(PC-SP)-128
1170 IFA<>133THEN1070
1180 FORI=0TO7
1190 BY(I)=0
1200 FORJ=0TO7
1210 BY(I)=BY(I)-2↑J*(PEEK(3448+7-J+40*I)=81)
1220 NEXTJ:NEXTI
1230 RETURN
2000 PRINTCHR$(147)
2010 PRINT"1 NORMALE":PRINT
2020 PRINT"2 REVERSATO":PRINT
2030 PRINT"3 SIMMETRIA VERTICALE":PRINT
2040 PRINT"4 SIMMETRIA ORIZZONTALE":PRINT
2050 PRINT"5 RUOTATO DI 90 GRADI ";
2055 PRINT"IN SENSO ORARIO":PRINT
2060 GETKEYA$
2070 A=VAL(A$):IFA=0ORA>5THEN2060
2080 INPUT"CARATTERE NUMERO ";NC%
2090 IFNC%>255ORN%<0THEN2080
2100 ONAGOSUB2200,2300,2400,2500,2600
2110 FORI=0TO7:POKE61440+8*NC%+I,BY(I):NEXT
2120 RETURN
2200 RETURN
2300 FORI=0TO7:BY(I)=255-BY(I):NEXT
2310 RETURN
```

```
2400 FORI=0TO7
2410 FORJ=0TO3
2420 BD=(BY(I)AND2↑J)/2↑J:BY(I)=BY(I)-2↑J*BD
2430 BS=(BY(I)AND2↑(7-J))/2↑(7-J)
2435 BY(I)=BY(I)-2↑(7-J)*BS
2440 BY(I)=BY(I)+2↑J*BS:BY(I)=BY(I)+2↑(7-J)*BD
2450 NEXTJ,I
2460 RETURN
2500 FORI=0TO3
2510 T=BY(I)
2520 BY(I)=BY(7-I)
2530 BY(7-I)=T
2540 NEXTI
2550 RETURN
2600 FORI=0TO7
2610 B2(I)=BY(I):BY(I)=0
2620 NEXTI
2630 FORI=0TO7
2640 FORJ=0TO7
2645 EX=(B2(I)AND2↑(7-J))/2↑(7-J)
2650 BY(J)=BY(J)OR(2↑(EX*I)*-(EX<>0))
2660 NEXTJ,I
2670 RETURN
3000 PRINTCHR$(147)
3010 INPUT"CARATTERE DA CORREGGERE ";NC%
3020 IFNC%>255ORN%<0THEN3010
3030 PRINTUS$
3040 FORI=1TO8:PRINTCHR$(17);:NEXTI
3050 FORI=0TO7
3060 FORJ=1TO16:PRINTCHR$(32);:NEXT
3070 FORJ=0TO7
3080 IFPEEK(61440+8*NC%+I)AND2↑(7-J)THENSI=-1
3085 IFSITHENPRINTCHR$(209);:SI=0:GOTO3100
3090 PRINTCHR$(43);
3100 NEXTJ:PRINT:NEXTI
3110 GOTO1060
4000 PRINTCHR$(147)"PREMI F1 PER TORNARE"
4005 PRINTCHR$(17)"PREMI I TASTI CORRISPONDENTI"
4006 PRINT"AI CARATTERI CHE VUOI VEDERE"
4010 FORI=1TO2000:NEXTI
4020 PRINTCHR$(147);
4030 POKE65298,PEEK(65298)AND251
4035 POKE65299,(PEEK(65299)AND3)+240
4040 GETKEYA$
4050 IFA$<>CHR$(133)THENPRINTA$;:GOTO4040
4060 POKE65298,PEEK(65298)OR4
4070 POKE65299,(PEEK(65299)AND3)+208
4080 COLOR1,7,5:RETURN
5000 PRINTCHR$(147)
5010 PRINT"DISCO O NASTRO ?"
5015 GETDV$:IFDV$<>"N"ANDDV$<>"D"THEN5015
5020 PRINT
5030 INPUT"SET NUMERO ";NS%
```

```
5040 PRINT
5050 PRINT"FINO A CHE CARATTERE VUOI SALVARE ";
5055 INPUTNC%
5060 IFNC%>255THEN5040
5070 IFDV$="N"THEN5500
5080 OPEN1,8,2,"@:SET #"+STR$(NS%)+",S,W"
5082 PRINT:PRINTDS$
5085 IFDSTHENFORI=1TO2000:NEXT:CLOSE1:RETURN
5090 PRINT#1,CHR$(NC%);:FORI=0TO(NC%+1)*8-1
5100 PRINT#1,CHR$(PEEK(61440+I));
5110 NEXTI
5120 CLOSE1
5130 RETURN
5500 OPEN1,1,1,"SET #"+STR$(NS%)
5510 PRINT#1,CHR$(NC%);:FORI=0TO(NC%+1)*8-1
5520 PRINT#1,CHR$(PEEK(61440+I));
5530 NEXTI
5540 CLOSE1
5550 RETURN
6000 PRINTCHR$(147)
6010 PRINT"DISCO O NASTRO ?"
6015 GETDV$:IFDV$<>"N"ANDDV$<>"D"THEN6015
6020 PRINT
6030 INPUT"SET NUMERO ";NS%
6070 IFDV$="N"THEN6500
6080 OPEN1,8,2,"SET #"+STR$(NS%)+",S,R"
6082 PRINT:PRINTDS$
6085 IFDSTHENFORI=1TO2000:NEXT:CLOSE1:RETURN
6090 GET#1,A$:NC%=ASC(A$):FORI=0TO(NC%+1)*8-1
6100 GET#1,A$:POKE61440+I,ASC(A$+CHR$(0))
6110 NEXTI
6120 CLOSE1
6130 RETURN
6500 OPEN1,1,0,"SET #"+STR$(NS%)
6510 GET#1,A$:NC%=ASC(A$):FORI=0TO(NC%+1)%*8-1
6520 GET#1,A$:POKE61440+I,ASC(A$+CHR$(0))
6530 NEXTI
6540 CLOSE1
6550 RETURN
7000 PRINTCHR$(17)"SICURO ?"
7010 GETKEYA$
7020 IFA$="S"THENSYS32768
7030 RETURN
```

Il programma GRAF5 puo' essere diviso in 14 sezioni ben distinte:

-1: 10-40: inizializza il calcolatore.

-2: 100-210: mostra il menu' e salta all'opzione richiesta.

-3: 1000-1230: ti permette di disegnare un nuovo carattere e riempie un vettore con i dati relativi al carattere disegnato.

-4: 2000-2120: mostra il menu' di memorizzazione dei caratteri, salta alla routine per il tipo di operazione richiesta sul vettore e memorizza il carattere.
-5: 2200: lascia il vettore invariato: e' stata messa per rispondere alla chiamata della linea 2100.
-6: 2300-2310: pone nel vettore il "negativo" del carattere contenuto prima.
-7: 2400-2460: pone nel vettore il carattere che gode di simmetria verticale rispetto a quello contenuto prima.
-8: 2500-2550: pone nel vettore il carattere che gode di simmetria orizzontale rispetto a quello contenuto prima.
-9: 2600-2670: pone nel vettore un carattere ruotato di 90 gradi in senso orario rispetto a quello contenuto prima. Chiamando 2 volte questa routine otterrai una rotazione di 180 gradi. Chiamandola 3 volte otterrai una rotazione di 90 gradi in senso antiorario.
-10: 3000-3110: riempie il vettore con i dati relativi al carattere che vuoi correggere e, saltando alla linea 1060, ti permette la correzione.
-11: 4000-4080: ti permette di vedere i caratteri che hai creato.
-12: 5000-5550: salva su disco o su nastro i caratteri che hai programmato.
-13: 6000-6550: carica da disco o da nastro un set salvato precedentemente.
-14: 7000-7030: chiude il programma.

COMMENTO A GRAF5 SEZIONE PER SEZIONE

SEZIONE 1
.10: schermo e sfondo neri, caratteri azzurri.
.15: set maiuscolo, disabilita la funzione di SHIFT-CBM.
.20: fine della memoria a 14336 ($3800).
.30: inizializza la costante US$.
.40: assegna a F1 il CHR$(133) per poterlo usare con l'istruzione GETKEY. Il tasto F1 serve per uscire dalle fasi del programma e tornare al menu' principale.

SEZIONE 2
.100/180: mostra le opzioni.
.190: accetta un tasto tra 1 e 9.
.195: pone nella variabile I il valore del tasto premuto.
.200: se I<7 salta alla routine richiesta.
.210: torna a mostrare le opzioni.

SEZIONE 3:
E' la sezione piu' complicata del programma: per comprenderne bene il funzionamento possiamo dividerla in 3 sottosezioni.

.A) . 1000/1010: disegna una griglia 8X8 su cui creare il carattere.

.B) . 1060/1170: permette di disegnare il carattere.

.C . 1180-1230: riempie il vettore BY(7) con i dati relativi al carattere disegnato sulla griglia.

Torniamo al commento linea per linea:

.1000: pulisce lo schermo e scrive in alto PREMI F1 PER USCIRE.

.1010: disegna la griglia.

.1060: pone nella variabile PC (posizione cursore) l'indirizzo del byte della mappa degli attributi corrispondente al "+" della griglia in alto a sinistra. Azzera le variabili XC e YC (coordinate X e Y del cursore).

.1070: pone a 1 il bit 7 del byte della mappa degli attributi corrispondente al cursore. In questo modo il carattere che corrisponde al cursore, lampeggia (vedi Paragrafo 4.8).

.1080: attende la pressione di un tasto.

.1090: A = codice ASCII ricevuto da tastiera e SP=0 (SP=spostamento che verra' effettuato).

.1100: se il carattere ricevuto e' "CURSORE A DESTRA" e il cursore non e' all'estrema destra della griglia (XC=7) allora incrementa XC e PC e pone SP=1.

.1110: se il carattere ricevuto e' "CURSORE A SINISTRA" e il cursore non e' all'estrema sinistra della griglia (XC=0) allora decrementa XC e PC e pone SP=-1.

.1120: se il carattere ricevuto e' "CURSORE IN BASSO" e il cursore non e' sull'ultima linea della griglia (YC=7) allora incrementa YC, somma 40 a PC e pone SP=40 (40 e' il numero di caratteri contenuti in una linea dello schermo).

.1130: se il carattere ricevuto e' "CURSORE IN ALTO" e il cursore non e' sulla prima linea della griglia (YC=0) allora decrementa YC, sottrae 40 a PC e pone SP=-40.

.1140: se il carattere ricevuto da tastiera e' SPAZIO pone nella posizione della mappa video corrispondente alla posizione del cursore una pallina per indicare che quel punto e' "acceso" (1024 e' la differenza tra l'indirizzi della mappa degli attributi e i corrispondenti indirizzi della mappa video).

.1150: se il carattere ricevuto da tastiera e' DELETE pone nella posizione della mappa video corrispondente alla posizione del cursore un "+" per indicare che quel punto e' "spento".

.1160: pone a 0 il bit 7 del byte della mappa degli attributi corrispondente alla vecchia posizione del cursore (PC-SP).

.1170: se non e' stato premuto F1 torna alla linea 1070 dove fa lampeggiare la posizione corrente del cursore.

.1180: per ogni riga della griglia esegue fino a 1220.

.1190: pone a 0 il corrispondente elemento del vettore.

.1200: per ogni elemento di una riga esegue fino a 1220

.1210: somma alla variabile corrispondente alla riga interessata 2 elevato alla posizione dell'elemento di riga considerato

(partendo da destra), se in tale posizione vi e' una pallina; altrimenti somma 0.
.1220: chiude il ciclo degli elementi e quello delle righe. A questo punto nel vettore BY sono contenuti gli 8 numeri necessari per programmare il carattere disegnato sulla griglia.
.1230: fine della routine.

SEZIONE 4
.2000/2055: mostra le opzioni relative alla memorizzazione dei caratteri.
.2060/2070: accetta dalla tastiera un numero tra 1 e 5 e lo pone nella variabile A.
.2080/2090: accetta un numero tra 0 e 255 e lo pone nella variabile NC% (il carattere che si vuole programmare).
.2100: salta alla routine di modifica del vettore richiesta.
.2110: memorizza il carattere ponendo in 8 byte di memoria a partire da 14336+8*NC% il contenuto del vettore.
.2120: torna dalla routine.

SEZIONE 5
.2200: e' stata messa per rispondere alla chiamata della linea 2100.

SEZIONE 6
.2300: pone in ogni elemento del vettore la differenza tra 255 e il valore contenuto prima. Questa operazione compiuta su numeri minori di 256 (come nel nostro caso), nega gli 8 bit del numero.
.2310: torna dalla routine.

SEZIONE 7
.2400: per ogni elemento del vettore esegue fino a 2450.
2410: per J da 0 a 3 esegue fino a 2450.
.2420: pone il valore del bit j-esimo dell'elemento considerato del vettore nella variabile BD (bit di destra) e lo azzera.
.2430/2435: pone il valore del bit (7-j)-esimo dell'elemento considerato del vettore nella variabile BS (bit di sinistra) e lo azzera.
.2440: pone il bit di sinistra al posto del bit di destra e viceversa.
.2450: chiude i due cicli.
.2460: torna dalla routine.

SEZIONE 8
.2500: per i primi 4 elementi del vettore esegue fino a 2540.
.2510: pone nella variabile T (temporanea) il valore dell'elemento considerato.
.2520: pone nell'elemento considerato il valore dell'elemento

simmetrico del vettore.
.2530: pone nell'elemento simmetrico il valore di T.
.2540: chiude il ciclo.
.2550: torna dalla routine.

SEZIONE 9
.2600/2620: trasferisce il vettore BY nel vettore B2 e lo azzera.
.2630: per ogni bit degli elementi del vettore esegue fino a 2660 (indice I).
.2640: per ogni elemento del vettore esegue fino a 2660 (indice J).
.2645/2650: pone il bit considerato dell'elemento in esame uguale al bit j-esimo dell'elemento simmetrico all'i-esimo.
.2660: chiude i due cicli.
.2670: torna dalla routine.

SEZIONE 10
.3000/3020: chiede il numero (D/CODE) del carattere da correggere. Accetta un numero tra 0 e 255 e lo pone nella variabile NC%.
.3030: pulisce lo schermo e scrive centrato in alto PREMI F1 PER USCIRE.
.3040: sposta il cursore piu' in basso di 4 linee.
.3050: per 8 volte esegue fino a 3100.
.3060: sposta il cursore a destra di 16 posizioni.
.3070/3100: scrive un "+" se il bit considerato contiene 0, una pallina altrimenti.
.3110: salta alla routine di programmazione del carattere.

SEZIONE 11
.4000/4006: fornisce le istruzioni all'utente.
.4010: attende circa 10 secondi.
.4020: cancella lo schermo.
.4030/4035: descrizione dei caratteri in RAM a partire da 14336 ($3800).
.4040: attende la pressione di un tasto.
.4050: se il tasto premuto non e' F1, scrive il carattere (che puo' essere anche un carattere di controllo del cursore o del colore) e torna ad attendere il prossimo.
.4060/4080: se e' stato premuto F1 riassume la ROM come descrizione dei caratteri, passa a scritte blu' e esce dalla routine.

SEZIONE 12
.5000: pulisce lo schermo.
.5010/5015: pone in DV$ la periferica scelta.
.5020/5030: pone in NS% il numero distintivo del set di caratteri.

.5040/5060: pone in NC% (D/CODE dell'ultimo carattere che si vuole salvare) un numero tra 0 e 255.
.5070: se la periferica scelta e' il nastro salta a 5500.
.5080/5085: apre il file su disco e torna se si verifica un errore.
.5090/5110: scrive su disco il numero di caratteri da salvare seguito dai dati relativi ai caratteri.
.5120/5130: chiude il file e torna dalla routine.
.5500: apre il file su nastro.
.5510/5530: scrive su nastro il numero di caratteri da salvare seguito dai dati relativi ai caratteri.
.5540/5550: chiude il file e torna dalla routine.

SEZIONE 13
.6000: pulisce lo schermo.
.6010/6015: pone in DV$ la periferica scelta.
.6020/6030: pone in NS% il numero distintivo del set di caratteri.
.6070: se la periferica scelta e' il nastro salta a 6500.
.6080/6085: apre il file su disco e torna se si verifica un errore.
.6090/6110: legge da disco il numero di caratteri da caricare e quindi carica i dati relativi ai caratteri.
.6120/6130: chiude il file e torna dalla routine.
.6500: apre il file su nastro.
.6510/6530: legge da nastro il numero di caratteri da caricare e quindi carica i dati relativi ai caratteri.
.6540/6550: chiude il file e torna dalla routine.

SEZIONE 14
.7000: porta il cursore piu' giu' di una linea e ti chiede se sei sicuro di voler abbandonare il programma.
.7010: attende la pressione di un tasto.
.7020: se la risposta e' "S" allora salta alla routine di inizializzazione del BASIC.
.7030: altrimenti torna dalla routine.

Con questo programma e' quindi possibile creare in memoria nuovi caratteri e salvarli su disco o cassetta. Se, in un programma che usa caratteri definiti, vuoi evitare di caricare i dati relativi ai caratteri da nastro, ma preferisci che siano gia' nel programma sotto forma di linee DATA, puoi usare il programma GRAF6, che converte i dati contenuti dal byte 14336 ($3800) in poi in linee DATA.

```
10 REM GRAF6
20 INPUT"FINO A CHE CARATTERE ";C%
50 PRINTCHR$(147);
```

```
60 N=1000+I*10:GOSUB160:PRINTN$"DATA";
70 FORJ=0TO7
80 N=PEEK(61440+I*8+J):GOSUB160:PRINTN$",";
90 NEXT
100 PRINTCHR$(20)
110 PRINT"I="I+1":C%="C%;
120 IFI<C%THENPRINT":GOTO50":GOTO140
130 PRINT":GOTO170"
140 POKE239,3:POKE1319,19:POKE1320,13
150 POKE1321,13:END
160 N$=STR$(N):N$=RIGHT$(N$,LEN(N$)-1):RETURN
170 POKE239,3:POKE1319,19:POKE1320,13:POKE1321,1
3:K=K+10:PRINTCHR$(147)K:PRINT"K="K":IFK<170THEN
170"
```

GRAF6 puo' essere usato anche per convertire in linee DATA programmi in linguaggio macchina o altri tipi di dati presenti in memoria. Il programma si basa sul fatto che il COMMODORE PLUS-4, appena esce da un programma, scrive i caratteri che si trovano nel buffer della tastiera (da 1319 a 1328 ($0527/$0530)) e si comporta esattamente come se fossero stati premuti dall'utente.

COMMENTO A GRAF6

.20: chiede fino a che carattere vuoi convertire in linee DATA. Il programma convertira' 8X(C%+1) byte a partire dall'indirizzo 14336 ($3800).

.50: pulisce lo schermo.

.60: il numero N che deve essere assegnato alla prossima linea DATA e' 1000+I*10, trasforma il numero N in una stringa (appoggiandosi ad una routine in 160), scrive una stringa che contiene N e la parola BASIC DATA. Puoi scegliere da quale linea partire, sostituendo a 1000 il numero di linea che preferisci, e il passo con cui incrementare il numero di linea, sostituendo a 10 il passo desiderato.

.70/90: per gli 8 byte di ogni carattere pone in N il contenuto del byte interessato, lo trasforma in una stringa utilizzando la stessa routine di prima, scrive la stringa e una virgola.

.100: cancella l'ultima virgola.

.110: scrive sul video le istruzioni BASIC che incrementeranno l'indice I e riporranno in memoria il valore di C% (le variabili vengono infatti perse ogni volta che si introduce una nuova linea di programma).

.120: se il carattere trattato non e' l'ultimo, la linea 120 scrive GOTO 50 e salta a 140.

.130: se il carattere trattato e' l'ultimo, scrive GOTO 170.

.140/150: pone 3 nel byte 239 (il byte 239 indica il numero di caratteri validi nel buffer di tastiera), pone nel buffer di tastiera il codice ASCII di HOME e due volte il codice ASCII di RETURN e esce dal programma. A questo punto il COMMODORE PLUS-4

trova nel buffer di tastiera HOME e quindi porta il cursore sulla linea DATA, un RETURN e quindi introduce nel programma la linea DATA portando il cursore sulla seconda linea scritta dal programma; trova il secondo RETURN ed esegue quindi i comandi scritti sulla linea, aggiorna cioe' le variabili e salta a 50 o a 170.

.160: e' la routine che trasforma il numero N nella corrispondente stringa N$ usando la funzione STR$ e scartando il carattere che contiene il segno.

.170: usando il metodo appena descritto, cancella le linee da 10 a 170.

9.5 CARATTERI A SFONDO PROGRAMMABILE

Tu sai che quando scrivi un carattere con il COMMODORE PLUS-4 puoi sceglierne il colore, selezionando il colore 1, usando i codici di controllo del colore del cursore, o scrivendo nella mappa degli attributi il codice desiderato. In questo modo selezioni il colore dello "inchiostro" con cui il COMMODORE PLUS-4 scrive il carattere, ma quello della "carta" (lo sfondo) rimane sempre lo stesso. TED ha la capacita' di leggere 4 diversi colori di sfondo quando e' posto in modo SFONDO PROGRAMMABILE. In questo modo, infatti, quando TED legge dalla mappa video il codice del carattere da visualizzare, considera i 6 bit meno significativi come codice del carattere, e i due bit rimanenti come codice del colore dello sfondo. Hai cosi' a disposizione un set di soli 64 (2 elevato a 6) caratteri, ma puoi scegliere per ciascuno di essi un colore di sfondo su 4 possibili. Puoi naturalmente scegliere questi 4 colori tra i 121 a disposizione. Con il modo a sfondo programmabile puoi ottenere delle bellissime maschere o dei coloratissimi quadri grafici. I caratteri che si perdono con il modo a sfondo programmabile sono:

. tutto il set MINUSCOLO/MAIUSCOLO (TED non considera, infatti, il bit 2 del registro 13, quando e' in modo SFONDO PROGRAMMABILE, non e' quindi possibile selezionare il set MINUSCOLO/MAIUSCOLO).

. tutti i caratteri in campo inverso.

. tutti i caratteri grafici.

Ti rimangono quindi a disposizione solo le lettere maiuscole, le dieci cifre, i segni di punteggiatura e delle operazioni matematiche.

Dei 4 colori-sfondo per i caratteri, uno e' quello dello schermo, i codici degli altri tre vanno posti nei registri del TED che rispondono agli indirizzi 65302, 65303, 65304 ($FF16, $FF17, $FF18). Se L e' il numero della luminosita' (tra 0 e 7) e C e' il numero del colore (tra 1 e 16), devi porre nei registri il numero L*16+C-1.

. il colore dello schermo viene dato come sfondo ai caratteri

che hanno nei due bit piu' significativi del codice 00.
. il colore del registro di indirizzo 65302 (colore di sfondo 1) viene dato come sfondo ai caratteri che hanno nei due bit piu' significativi del codice 01.
. il colore del registro di indirizzo 65303 (colore di sfondo 2) viene dato come sfondo ai caratteri che hanno nei due bit piu' significativi del codice 10.
. il colore del registro di indirizzo 65304 (colore di sfondo 3) viene dato come sfondo ai caratteri che hanno nei due bit piu' significativi del codice 11.
Ricorda che, normalmente, il bit 7 vale 1 per i caratteri in campo inverso: nel modo a sfondo programmabile, quindi, se premi una A ottieni una A su sfondo normale, mentre RVS ON + A produce una A con colore di sfondo 2. Per le lettere e lo spazio vale anche la regola che SHIFT-LETTERA visualizza la lettera con colore di sfondo 1 e RVS ON + SHIFT-LETTERA visualizza la lettera con colore di sfondo 3. Questa regola non vale per i numeri e i rimanenti simboli. Alla fine del paragrafo riportiamo la Tabella 9.1; essa ti aiuta a ottenere facilmente i 64 simboli con i vari colori di sfondo. Per entrare in modo SFONDO PROGRAMMABILE basta porre a 1 il bit 6 del registro 6 del TED che risponde all'indirizzo 65286 ($FF06). L'istruzione da eseguire e' quindi la seguente:

```
POKE 65286, PEEK(65286) OR 64
```

mentre l'istruzione da eseguire per tornare al modo modo normale e':

```
POKE 65286, PEEK(65286) AND 191
```

Come prova di programmazione dello sfondo dei caratteri segue il programma GRAF7.

```
0 rem graf7
10 color0,1:color4,1
20 poke65286,peek(65286)or64
30 poke65302,92
40 poke65303,94
50 poke65304,82
60 printchr$(147)
70 color1,8,5
80 char1,11,6,"commodore plus 4"
90 color1,6,3
100 char1,9,9,"                      "
110 char1,9,10," PROGRAMMA DI PROVA "
120 char1,9,11,"                      "
130 color1,7,3
140 char1,13,13,chr$(18)+"            "
150 char1,13,14,chr$(18)+" caratteri "
```

```
160 char1,13,15,chr$(18)+"           "
170 color1,3,3
180 char1,7,17,chr$(18)+"
"
190 char1,7,18,chr$(18)+" A SFONDO PROGRAMMABILE
"
200 char1,7,19,chr$(18)+"
"
210 getkeya$
220 poke65286,peek(65286)and191
```

COMMENTO A GRAF7

.10: schermo e bordo neri.
.20: entra in modo sfondo programmabile.
.30: colore di sfondo 1: colore 13 con luminosita' 5 (5*16+13¬1=92).
.40: colore di sfondo 1: colore 15 con luminosita' 5 (5*16+15¬1=94).
.30: colore di sfondo 1: colore 3 con luminosita' 5 (5*16+3¬1=82).
.60: pulisce lo schermo.
.70: colore delle scritte giallo.
.80: scrive in colonna 13, riga 6, "COMMODORE PLUS-4".
.90: colore delle scritte verde.
.100: scrive in colonna 9, riga 9, 20 spazi shiftati.
.110: scrive in colonna 9, riga 10, la scritta shiftata " PROGRAMMA DI PROVA ".
.120: scrive in colonna 9, riga 11, 20 spazi shiftati.
.130: colore delle scritte blu.
.140: scrive in colonna 13, riga 13, 11 spazi reversati.
.150: scrive in colonna 13, riga 14, la scritta reversata " CARATTERI ".
.160: scrive in colonna 13, riga 15, 11 spazi reversati.
.170: colore delle scritte rosso.
.180: scrive in colonna 7, riga 17, 24 spazi shiftati e reversati.
.190: scrive in colonna 7, riga 18, la scritta shiftata e reversata " A SFONDO PROGRAMMABILE ".
.200: scrive in colonna 13; riga 15, 24 spazi shiftati e reversati.
.210: attende la pressione di un tasto.
.220: torna al modo normale.

Nota che in modo sfondo programmabile, il bit 7 dell'attributo viene ignorato, non e' possibile quindi ottenere caratteri lampeggianti (effetto FLASH). Ti proponiamo ora il programma

GRAF8, leggermente piu' complicato, che usa il modo sfondo programmabile con i caratteri definiti in RAM: in questo esempio gli stessi caratteri vengono "disegnati" su sfondi diversi.

```
0 REM GRAF8
10 POKE56,240:POKE55,0:CLR:DM=1024
20 FORI=61440TO61631
30 READA
40 POKEI,A
50 NEXT
60 FORI=0TO7:POKE61696+I,0:NEXT
70 PRINTCHR$(147)
80 COLOR0,14,6:COLOR4,2,5
90 POKE65302,85
100 POKE65303,73
110 POKE65304,103
120 POKE65299,(PEEK(65299)AND3)+240
121 POKE65298,PEEK(65298)AND251
130 POKE65286,PEEK(65286)OR64
140 COLOR1,3,3:CHAR1,39,7,"V"
150 FORI=0TO2
160 CHAR1,38-I,8+I,LEFT$("VWWW",I+2)
170 NEXT
180 FORI=3592TO3751:POKEI,96:NEXT
190 FORI=3752TO3911:POKEI,160:NEXT
200 FORI=3912TO4071:POKEI,96:NEXT
210 FORI=0TO2:FORJ=0TO2:POKE3549+J+I*40,224
220 NEXT:NEXT
230 OG=3588:T=1:GOSUB500
240 OG=3678:T=0:GOSUB500
250 OG=3538:T=0:GOSUB500
260 OG=3848:T=1:GOSUB500
270 GETKEYA$
280 POKE65286,PEEK(65286)AND191
290 POKE65299,(PEEK(65299)AND3)+208
300 POKE65298,PEEK(65298)OR4
310 PRINTCHR$(147):END
500 IFTTHEN620
510 POKEOG,(PEEK(OG)AND196)+0
511 POKEOG-DM,113:OG=OG+1
520 POKEOG,(PEEK(OG)AND196)+1
521 POKEOG-DM,113:OG=OG+39
530 POKEOG,(PEEK(OG)AND196)+2
531 POKEOG-DM,54:OG=OG+1
540 POKEOG,(PEEK(OG)AND196)+3
541 POKEOG-DM,54:OG=OG+38
550 POKEOG,(PEEK(OG)AND196)+6
551 POKEOG-DM,54:OG=OG+1
560 POKEOG,(PEEK(OG)AND196)+4
561 POKEOG-DM,54:OG=OG+1
570 POKEOG,(PEEK(OG)AND196)+5
571 POKEOG-DM,54:OG=OG+39
580 POKEOG,(PEEK(OG)AND196)+7
```

```
581 POKEOG-DM,113:OG=OG+1
590 POKEOG,(PEEK(OG)AND196)+8
591 POKEOG-DM,113:OG=OG+39
600 POKEOG,(PEEK(OG)AND196)+9
601 POKEOG-DM,113:OG=OG+1
610 POKEOG,(PEEK(OG)AND196)+10
611 POKEOG-DM,113:RETURN
620 POKEOG,(PEEK(OG)AND196)+12
621 POKEOG-DM,113:OG=OG+1
630 POKEOG,(PEEK(OG)AND196)+11
631 POKEOG-DM,113:OG=OG+39
640 POKEOG,(PEEK(OG)AND196)+14
641 POKEOG-DM,54:OG=OG+1
650 POKEOG,(PEEK(OG)AND196)+13
651 POKEOG-DM,54:OG=OG+39
660 POKEOG,(PEEK(OG)AND196)+16
661 POKEOG-DM,54:OG=OG+1
670 POKEOG,(PEEK(OG)AND196)+15
671 POKEOG-DM,54:OG=OG+1
680 POKEOG,(PEEK(OG)AND196)+17
681 POKEOG-DM,54:OG=OG+38
690 POKEOG,(PEEK(OG)AND196)+19
691 POKEOG-DM,113:OG=OG+1
700 POKEOG,(PEEK(OG)AND196)+18
701 POKEOG-DM,113:OG=OG+39
710 POKEOG,(PEEK(OG)AND196)+21
711 POKEOG-DM,113:OG=OG+1
720 POKEOG,(PEEK(OG)AND196)+20
721 POKEOG-DM,113:RETURN
730 DATA0,0,1,3,7,15,31,31
740 DATA0,96,240,240,224,192,192,224
750 DATA31,63,63,127,127,63,31,7
760 DATA224,54,191,255,246,240,224,0
770 DATA7,31,62,125,251,247,207,159
780 DATA0,176,216,220,238,239,231,195
790 DATA0,0,0,0,1,7,7,15
800 DATA31,31,15,23,27,29,30,60
810 DATA192,128,128,225,243,119,127,62
820 DATA126,255,126,0,0,0,0,0
830 DATA28,0,0,0,0,0,0,0
840 DATA0,0,128,192,224,240,248,248
850 DATA0,6,15,15,7,3,3,7
860 DATA248,252,252,254,254,252,248,224
870 DATA7,108,253,255,111,15,7,0
880 DATA224,248,124,190,223,239,243,249
890 DATA0,13,27,59,119,247,231,195
900 DATA0,0,0,0,128,224,224,240
910 DATA248,248,240,232,216,184,120,60
920 DATA3,1,1,135,207,238,254,124
930 DATA126,255,126,0,0,0,0,0
940 DATA56,0,0,0,0,0,0,0
950 DATA1,3,7,15,31,63,127,255
960 DATA255,255,255,255,255,255,255,255
```

GRAF8 memorizza i dati relativi a 22 caratteri necessari a disegnare dei puffi e, grazie a una subroutine, li pone in un qualsiasi punto di uno schermo variopinto.

COMMENTO A GRAF8

.10: pone la fine della memoria a 61440 ($F000) per proteggere i caratteri dalle variabili e pone nella variabile DM (Differenza Mappe) la differenza tra il primo indirizzo della mappa degli attributi e il primo indirizzo della mappa video.

.20/50: pone in memoria i dati delle immagini dei caratteri.

.60: azzera il carattere 32 (lo spazio).

.70: pulisce lo schermo.

.80: schermo blu chiaro e bordo grigio.

.90/110: selezionano i tre colori di sfondo:
1=verde, 2=marrone, 3=giallo.

.120: descrizione dei caratteri a partire da 61440 ($F000).

.130: entra in modo sfondo programmabile.

.140/170: disegna un triangolo rosso (che sara' il tetto di una casetta) con i caratteri programmati. Nota che non puoi sovrapporre dei puffi al tetto perche' questo e' stato ottenuto con dei caratteri e non con degli spazi colorati.

.180: disegna una striscia di spazi shiftati (colore di sfondo 1).

.190: disegna una striscia di spazi reversati (colore di sfondo 2).

.200: disegna un'altra striscia di spazi shiftati (colore di sfondo 1).

.210/220: disegna un quadrato di 3X3 spazi shiftati e reversati (colore di sfondo 3) che sara' la casetta.

.230: disegna un puffo di tipo 1 (che guarda a sinistra), il cui angolo in alto a sinistra verra' posto nel byte di memoria video 3588, usando la routine in 500.

.240: disegna un puffo di tipo 0 (che guarda a destra), il cui angolo in alto a sinistra verra' posto nel byte di memoria video 3678, usando la routine in 500.

.250: disegna un puffo di tipo 0 (che guarda a destra), il cui angolo in alto a sinistra verra' posto nel byte di memoria video 3538, usando la routine in 500.

.260: disegna un puffo di tipo 1 (che guarda a sinistra), il cui angolo in alto a sinistra verra' posto nel byte di memoria video 3848, usando la routine in 500.

.270: attende che sia premuto un tasto.

.280: esce dal modo a sfondo programmabile.

.290/300: descrizione dei caratteri in ROM.

.310: pulisce lo schermo ed esce.

.500: inizia la routine che disegna i puffi. Salta se T<>0 cioe' se il puffo e' di tipo 1.

.510/610: ogni linea disegna uno degli 11 caratteri che formano un puffo di tipo 0 e pone nel corrispondente byte della mappa degli attributi il colore adatto. Per disegnare un carattere in un byte facciamo la AND tra il contenuto del byte e 196 per ottenere il valore dei bit 7 e 6, sommiamo quindi il codice del carattere e poniamo il risultato nel byte. In questo modo lasciamo inalterato il colore dello sfondo.

.620/720: ogni linea disegna uno degli 11 caratteri che formano un puffo di tipo 1 e pone nel corrispondente byte della mappa degli attributi il colore adatto.

.730/960: dati relativi ai caratteri dei puffi e del tetto della casa.

TABELLA 9.1				
SIMB.	TASTI DA PREMERE			
	SFONDO	COLORE 1	COLORE 2	COLORE 3
@	@	SHIFT+*	RVS+@	RVS+SHIFT+*
A	A	SHIFT+A	RVS+A	RVS+SHIFT+A
B	B	SHIFT+B	RVS+B	RVS+SHIFT+B
C	C	SHIFT+C	RVS+C	RVS+SHIFT+C
D	D	SHIFT+D	RVS+D	RVS+SHIFT+D
E	E	SHIFT+E	RVS+E	RVS+SHIFT+E
F	F	SHIFT+F	RVS+F	RVS+SHIFT+F
G	G	SHIFT+G	RVS+G	RVS+SHIFT+G
H	H	SHIFT+H	RVS+H	RVS+SHIFT+H
I	I	SHIFT+I	RVS+I	RVS+SHIFT+I
J	J	SHIFT+J	RVS+J	RVS+SHIFT+J
K	K	SHIFT+K	RVS+K	RVS+SHIFT+K
L	L	SHIFT+L	RVS+L	RVS+SHIFT+L
M	M	SHIFT+M	RVS+M	RVS+SHIFT+M
N	N	SHIFT+N	RVS+N	RVS+SHIFT+N
O	O	SHIFT+O	RVS+O	RVS+SHIFT+O
P	P	SHIFT+P	RVS+P	RVS+SHIFT+P
Q	Q	SHIFT+Q	RVS+Q	RVS+SHIFT+Q
R	R	SHIFT+R	RVS+R	RVS+SHIFT+R
S	S	SHIFT+S	RVS+S	RVS+SHIFT+S
T	T	SHIFT+T	RVS+T	RVS+SHIFT+T
U	U	SHIFT+U	RVS+U	RVS+SHIFT+U
V	V	SHIFT+V	RVS+V	RVS+SHIFT+V
W	W	SHIFT+W	RVS+W	RVS+SHIFT+W
X	X	SHIFT+X	RVS+X	RVS+SHIFT+X

TABELLA 9.1				
SIMB.	TASTI DA PREMERE			
	SFONDO	COLORE 1	COLORE 2	COLORE 3
Y	Y	SHIFT+Y	RVS+Y	RVS+SHIFT+Y
Z	Z	SHIFT+Z	RVS+Z	RVS+SHIFT+Z
[	[	SHIFT++	RVS+[	RVS+SHIFT++
£	£	CBM+-	RVS+£	RVS+CBM+-
]	]	SHIFT+-	RVS+]	RVS+SHIFT+-
↑	↑	CBM+=	RVS+↑	RVS+CBM+=
←	←	CBM+*	RVS+←	RVS+CBM+*
SP	SP	SHIFT+SP	RVS+SP	RVS+SHIFT+SP
!	!	CBM+K	RVS+!	RVS+CBM+K
"	"	CBM+I	RVS+"	RVS+CBM+I
#	#	CBM+T	RVS+#	RVS+CBM+T
$	$	CBM+@	RVS+$	RVS+CBM+@
%	%	CBM+G	RVS+%	RVS+CBM+G
&	&	CBM++	RVS+&	RVS+CBM++
'	'	CBM+M	RVS+'	RVS+CBM+M
(	(	CBM+£	RVS+(	RVS+CBM+£
)	)	SHIFT+£	RVS+)	RVS+SHIFT+£
*	*	CBM+N	RVS+*	RVS+CBM+N
+	+	CBM+Q	RVS++	RVS+CBM+Q
,	,	CBM+D	RVS+,	RVS+CBM+D
-	-	CBM+Z	RVS+-	RVS+CBM+Z
.	.	CBM+S	RVS+.	RVS+CBM+S
/	/	CBM+P	RVS+/	RVS+CBM+P
0	0	CBM+A	RVS+0	RVS+CBM+A
1	1	CBM+E	RVS+1	RVS+CBM+E
2	2	CBM+R	RVS+2	RVS+CBM+R
3	3	CBM+W	RVS+3	RVS+CBM+W
4	4	CBM+H	RVS+4	RVS+CBM+H
5	5	CBM+J	RVS+5	RVS+CBM+J
6	6	CBM+L	RVS+6	RVS+CBM+L
7	7	CBM+Y	RVS+7	RVS+CBM+Y
8	8	CBM+U	RVS+8	RVS+CBM+U
9	9	CBM+O	RVS+9	RVS+CBM+O
:	:	SHIFT+@	RVS+:	RVS+SHIFT+@
;	;	CBM+F	RVS+;	RVS+CBM+F

TABELLA 9.1				
SIMB.	TASTI DA PREMERE			
	SFONDO	COLORE 1	COLORE 2	COLORE 3
<	<	CBM+C	RVS+<	RVS+CBM+C
=	=	CBM+X	RVS+=	RVS+CBM+X
>	>	CBM+V	RVS+>	RVS+CBM+V
?	?	CBM+B	RVS+?	RVS+CBM+B

9.6 CARATTERI MULTICOLORE

Questo paragrafo tratta dell'ultimo modo che hai a disposizione per rappresentare i caratteri con il COMMODORE PLUS-4: il modo MULTICOLORE. Come nei modi grafici 3 o 4, nel modo multicolore, ad ogni puntino che forma un carattere corrispondono 2 bit. In questo modo puoi associare ad ogni punto un colore scelto tra 4 possibili (invece che tra 2) ma la matrice di ogni carattere e' 4x8 (anziche' 8x8). Per entrare in modo MULTICOLORE devi porre a 1 il bit 4 del registro 22 del TED (indirizzo 65287 ($FF16)); bisogna cioe' eseguire l'istruzione:

```
POKE 65287, PEEK(65287) OR 16
```

per tornare in modo ad alta risoluzione devi usare l'istruzione:

```
POKE 65287, PEEK(65287) AND 239.
```

Per chiarire le idee vediamo come TED legge il carattere A in modo multicolore sapendo che gli 8 byte che descrivono la A contengono:

```
0 0 0 1 1 0 0 0
0 0 1 1 1 1 0 0
0 1 1 0 0 1 1 0
0 1 1 1 1 1 1 0
0 1 1 0 0 1 1 0
0 1 1 0 0 1 1 0
0 1 1 0 0 1 1 0
0 0 0 0 0 0 0 0
```

TED in modo multicolore interpreta ogni byte come una serie di quattro punti, cosi':

```
00 01 10 00   cioe' come:   A B C A
00 11 11 00                 A D D A
```

```
01 10 01 10        B C B C
01 11 11 10        B D D C
01 10 01 10        B C B C
01 10 01 10        B C B C
01 10 01 10        B C B C
00 00 00 00        A A A A
```

dove A, B, C e D sono rispettivamente:

- il colore dello schermo.
- il colore 3.
- il colore indicato dal registro 23 (indirizzo 65303 ($FF17)) con la convenzione 16*L+C-1, dove C e' il codice del colore e L la luminosita'.
- il colore proprio del carattere.

I primi 3 colori possono essere scelti tra i 16 del COMMODORE PLUS-4, mentre il quarto puo' essere solo uno dei primi 8. Questa piccola limitazione permette, in compenso, di visualizzare contemporaneamente caratteri multicolore e alta risoluzione. TED, infatti, visualizza il carattere come multicolore solo se il byte corrispondente della mappa degli attributi ha il bit 3 a 1 e assume come codice del colore il numero formato dai 3 bit meno significativi. Quando scegli il colore proprio del carattere dovrai sceglierlo tra i primi 8 e aggiungere 8 se lo vuoi multicolore o 0 se lo vuoi normale.
Ecco GRAF9 come esempio di programmazione di un carattere multicolore.

```
0 REM GRAF9
10 PRINTCHR$(147)
20 FORI=0TO7:READA:POKE61440+I,A:NEXT
30 FORI=0TO7:POKE61696+32*8+I,0:NEXT
40 COLOR0,15,0:COLOR4,15,0
50 COLOR1,14,4
60 POKE65303,16*7+1
70 COLOR3,3,3
80 POKE65287,PEEK(65287)OR16
90 POKE65299,(PEEK(65299)AND3)+240
100 POKE65298,PEEK(65298)AND251
110 PRINT"@"
120 GETKEYA$
130 POKE65287,PEEK(65287)AND239
140 POKE65299,(PEEK(65299)AND3)+208
150 POKE65298,PEEK(65298)OR4
1000 DATA165,165,165,165,240,240,240,240
```

COMMENTO A GRAF9

.10: pulisce lo schermo.

.20: pone in memoria i dati relativi al carattere di D/CODE 0.
.30: azzera il carattere di D/CODE 32 (lo spazio).
.40: schermo e bordo blu.
.50: colore del cursore 14 (in modo multicolore vuol dire colore D=6 (verde), carattere da visualizzare in modo multicolore).
.60: colore C=2 (bianco).
.70: colore B=3 (rosso).
.80: entra in modo multicolore.
.90/100: descrizione dei caratteri in 61440 ($F000).
.110: scrive il carattere di D/CODE 0.
.120: attende la pressione di un tasto.
.130: torna al modo alta risoluzione.
.140/150: caratteri in ROM.
.1000: dati relativi al carattere.

Il carattere che abbiamo programmato e' questo:

165 = 10100101 = CCBB
165 = 10100101 = CCBB
165 = 10100101 = CCBB
165 = 10100101 = CCBB
240 = 11110000 = DDAA
240 = 11110000 = DDAA
240 = 11110000 = DDAA
240 = 11110000 = DDAA

In questo modo il carattere programmato e' composto da 4 quadratini di colore diverso. Questo esempio mostra come TED possa porre 4 colori diversi nella posizione di un solo carattere, ma il risultato non e' certamente dei piu' pittoreschi. Prova quindi il programma GRAF10 il cui scopo e' quello di disegnare dei coloratissimi omini spaziali.

```
0 REM GRAF10
10 PRINTCHR$(147)
20 FORI=0TO47:READA:POKE61440+I,A:NEXT
30 FORI=0TO7:POKE61440+32*8+I,0:NEXT
40 COLOR0,1:COLOR4,1
50 COLOR1,16,6
60 POKE65303,16*4+6
70 COLOR3,3,3
80 POKE65287,PEEK(65287)OR16
90 POKE65299,(PEEK(65299)AND3)+240
100 POKE65298,PEEK(65298)AND251
110 CHAR,0,10," @A @A @A @A @A @A @A"
115 PRINT" @A @A @A @A @A @A"
120 PRINT" BC BC BC BC BC BC BC";
125 PRINT" BC BC BC BC BC BC"
```

```
130 PRINT" DE DE DE DE DE DE DE";
135 PRINT" DE DE DE DE DE DE"
140 GETKEYA$
150 POKE65287,PEEK(65287)AND239
160 POKE65299,(PEEK(65299)AND3)+208
170 POKE65298,PEEK(65298)OR4
1000 DATA192,192,49,53,21,81,65,85
1010 DATA3,3,76,92,84,69,65,85
1020 DATA84,85,22,5,3,5,21,85
1030 DATA21,85,148,80,192,80,84,85
1040 DATA85,85,21,5,1,34,42,10
1050 DATA85,85,84,80,64,136,168,160
```

COMMENTO A GRAF10

.10: pulisce lo schermo.
.20: programma i caratteri che formano l'omino.
.30: programma lo spazio.
.40: schermo e bordo neri.
.50: pone a 16 il colore dei caratteri (il colore D); saranno quindi caratteri multicolore con colore 8, cioe' giallo.
.60: pone a 7 (blu) il colore C.
.70: pone a 3 (rosso) il colore B.
.80: entra in modo multicolore.
.90/100: descrizione dei caratteri in 61440 ($F000).
.110/135: scrive partendo dalla colonna 0, riga 10, i caratteri che compongono gli omini.
.140: attende la pressione di un tasto.
.150: torna al modo normale.
.160/170: riporta la descrizione dei caratteri in ROM.
.1000/1050: dati relativi agli omini.

9.7 ANNULLAMENTO DELLO SCHERMO

E' possibile annullare tutto cio' che compare sullo schermo ponendo a 0 il bit 4 del registro 6 del TED. Tale registro risponde all'indirizzo 65286 ($FF06); l'istruzione da impartire e' quindi:

```
POKE 65286, PEEK(65286) AND 239
```

eseguendola lo schermo diventa completamente vuoto e dello stesso colore del bordo, come quando il calcolatore accede a nastro. Per riportare la situazione alla normalita' devi compiere l'operazione inversa, cioe' eseguire:

```
POKE 65286, PEEK(65286) OR 19
```

Puoi sfruttare questa opzione di TED per compiere delle operazioni sullo schermo mentre questo non si puo' vedere, ma il vero significato di questo bit e' un altro. Come sai TED chiede del tempo alla CPU per poter leggere dalla memoria i dati che gli servono per sapere cosa visualizzare. Quando lo schermo viene annullato, TED non ha piu' bisogno di quei dati e quindi "lascia in pace" la CPU che puo' svolgere le sue funzioni con una velocita' notevolmente piu' alta. Prova ora a far girare il programma GRAF11:

```
0 REM GRAF11
10 POKE65286,PEEK(65286)AND239
20 TI$="000000"
30 FORI=1TO10000:NEXT
40 A=INT(TI/6)/10
50 POKE65286,PEEK(65286)OR16
60 TI$="000000"
70 FORI=1TO10000:NEXT
80 B=INT(TI/6)/10
90 PRINT"CON SCHERMO ANNULLATO :"A"SECONDI"
100 PRINT"CON SCHERMO NON ANNULLATO :"B"SECONDI"
```

Come vedi il tempo impiegato a schermo annullato e' sensibilmente minore di quello impiegato nello stato normale. Ma il vantaggio piu' importante non e' nella velocita' che si guadagna, ma nella precisione con cui la CPU puo' calcolare i ritardi. Con adatte routine in linguaggio macchina, infatti, si possono calcolare ritardi con una precisione di meno di 1 milionesimo di secondo a patto di mascherare gli interrupt e di annullare lo schermo. Ritardi cosi' precisi servono in alcune operazioni di I/O come quelle del nastro.

9.8 SCORRIMENTO FINE (SMOOTH SCROLLING) E REGISTRO DI LINEA

TED permette di far scorrere tutto il contenuto dello schermo di un solo punto (un ottavo di carattere)., sia in direzione orizzontale che in direzione verticale.
I registri con cui si controllano queste operazioni sono rispettivamente i registri 7 e 6 che rispondono agli indirizzi 65287 ($FF07) e 65286 ($FF06). I bit 2-0 di questi registri indicano quale delle 8 possibili posizioni (da 0 a 7) devono assumere i caratteri sul video, mentre il bit 3 seleziona, se posto a 0, rispettivamente il modo a 38 colonne (1 per 40 colonne) e a 24 righe (1 per 25 righe).
Nota che lo schermo rimane sempre di 25 righe di 40 colonne, ma ponendo a 0 questi bit vengono visualizzate solamente 24 righe e 38 colonne: questo permette di aggiungere, nella parte nascosta dello schermo, i nuovi dati e farli entrare lentamente nella parte visibile realizzando lo scorrimento fine. Per ottenere uno scorrimento fine, quindi, devi:

. diminuire la parte visibile dello schermo nella direzione in cui lo vuoi far scorrere (ad esempio, se vuoi far scorrere lo schermo in direzione verticale, lo dovrai portare a 24 righe).
. porre i 3 bit meno significativi del registro interessato a 7 o a 0 (se lo scorrimento viene fatto dall'alto verso il basso a 0, viceversa a 7; da sinistra a destra a 0, viceversa a 7).
. scrivere i nuovi dati da visualizzare nella parte nascosta dello schermo.
. incrementare (o decrementare) il contenuto dei bit 2-1 del registro interessato fino al valore massimo (o minimo).
. riportare il contenuto dei bit 2-0 al valore del secondo passo e far scorrere tutta la scritta di un carattere intero nella direzione dello scorrimento. Questa operazione deve essere fatta con una routine in linguaggio macchina poiche' in BASIC si vede nettamente che questa operazione equivale a fare 7 passi indietro piu' 8 in avanti, mentre deve sembrare che sia un solo passo in avanti.
. tornare al punto 3.

Vediamo un paio di esempi che chiariscono meglio questi concetti:

```
0 REM GRAF12
10 COLOR0,15,1:COLOR1,14,6:COLOR4,15,1
20 PRINTCHR$(147)
30 PRINT"CHE SCRITTA VUOI FAR SCORRERE ":PRINT
40 GETKEYA$:IFA$=CHR$(13)ORLEN(B$)=253THEN60
50 B$=B$+A$:PRINTA$;:GOTO40
60 B$=B$+"  "
70 L=LEN(B$):PRINTCHR$(147)
80 POKE65287,PEEK(65287)AND247
90 FORI=1TOLEN(B$)
100 PRINTCHR$(19)" ";
110 PRINTCHR$(20);:POKE65287,(PEEK(65287)AND248)
+7
120 PRINTCHR$(17)CHR$(157)MID$(B$,I,1)
130 FORJ=6TO0STEP-1
140 POKE65287,(PEEK(65287)AND248)+J
150 NEXT J
160 NEXT I
170 GOTO90
```

COMMENTO A GRAF12

.10/30: inizializza il video.
.40/50: riceve una stringa che puo' essere lunga fino a 253 caratteri.
.60: aggiunge 2 spazi alla stringa.
.70/80: calcola la lunghezza della stringa, pulisce lo schermo e

seleziona il modo a 38 colonne.
.90: inizializza un ciclo di tanti passi quanti sono i caratteri che formano la stringa.
.100: posiziona il cursore sulla seconda colonna della prima riga.
.110: cancella un carattere in modo da far scorrere tutta la prima riga a sinistra e riportare il cursore nell'angolo in alto a sinistra e pone 7 nei bit 2-0 del registro dello scorrimento orizzontale (7 = massimo a destra).
.120: sposta il cursore in basso e quindi a sinistra per posizionarlo sull'ultima colonna della prima riga dove stampa l'I-esimo carattere della stringa.
.130/150: ciclo che decrementa il contenuto dei bit 2-0 del registro dello scorrimento orizzontale fino a 0, spostando le scritte gradualmente a sinistra.
.160: prossimo carattere della stringa.
.170: torna alla linea 90 per cominciare nuovamente dall'inizio della stringa.

Facendo girare GRAF12 ti accorgerai che un programma completamente BASIC non da' dei buoni risultati. La linea 110 andra' quindi sostituita con un'adeguata routine in linguaggio macchina come nel programma GRAF13 in cui la linea 110 e' stata sostituita dalla routine GRAF14. GRAF14 usa un registro di TED che non abbiamo ancora illustrato: il REGISTRO DI LINEA. Questo registro indica in ogni istante quale delle 313 linee che formano lo schermo viene disegnata da TED; risponde all'indirizzo 65309 ($FF1D) per gli 8 bit bassi e da' il bit 8 (servono infatti 9 bit, da 0 a 8, per esprimere un numero tra 0 e 312) nel bit meno significativo se l'indirizzo e' 65308 ($FF1C). Le linee che formano la parte in cui TED puo' visualizzare i caratteri vanno dalla 4 alla 203. GRAF14, prima di eseguire cio' che veniva svolto dalla linea 110 in GRAF12, controlla che il registro di linea contenga $CC (204) cioe' si assicura di svolgere le sue funzioni mentre TED sta disegnando la parte inferiore del bordo. Questo accorgimento e' necessario perche', anche se la routine in linguaggio macchina e' velocissima, provocherebbe gli stessi effetti del BASIC se svolgesse le sue funzioni mentre TED sta disegnando le linee 4-11, su cui essa agisce.
Vediamo ora i listati di GRAF13 e GRAF14.

```
0 REM GRAF13
5 POKE56,127:POKE55,233:CLR
6 FORI=0TO22:READA:POKE32745+I,A:NEXT
10 COLOR0,15,1:COLOR1,14,6:COLOR4,15,1
20 PRINTCHR$(147)
30 PRINT"CHE SCRITTA VUOI FAR SCORRERE ":PRINT
40 GETKEYA$:IFA$=CHR$(13)ORLEN(B$)=253THEN60
```

```
50 B$=B$+A$:PRINTA$;:GOTO40
60 B$=B$+"  "
70 L=LEN(B$):PRINTCHR$(147)
80 POKE65287,PEEK(65287)AND247
90 FORI=1TOLEN(B$)
100 PRINTCHR$(19)" ";
110 SYS32745
120 PRINTCHR$(17)CHR$(157)MID$(B$,I,1)
130 FORJ=6TO0STEP-1:FORK=1TO20:NEXT
140 POKE65287,(PEEK(65287)AND248)+J
150 NEXT J
160 NEXT I
170 GOTO90
180 DATA120,173,29,255,201,204,208,249
190 DATA169,20,32,210,255,173,7,255
200 DATA9,7,141,7,255,88,96
```

```
MONITOR
   PC  SR AC XR YR SP
; 0000 00 00 00 00 F8
. 7FE9  78          SEI
. 7FEA  AD 1D FF    LDA $FF1D
. 7FED  C9 CC       CMP #$CC
. 7FEF  D0 F9       BNE $7FEA
. 7FF1  A9 14       LDA #$14
. 7FF3  20 D2 FF    JSR $FFD2
. 7FF6  AD 07 FF    LDA $FF07
. 7FF9  09 07       ORA #$07
. 7FFB  8D 07 FF    STA $FF07
. 7FFE  58          CLI
. 7FFF  60          RTS
```

Il programma GRAF13 e' identico a GRAF12 tranne nelle linee:
.5: che abbassa i puntatori di memoria e pone la routine GRAF14 da 32745 ($7FE9) a 32767 ($7FFF).
.110: che salta alla routine in linguaggio macchina.

Abbiamo posto la routine GRAF14 in questi indirizzi perche' sono i piu' alti in cui la RAM e' presente anche quando e' selezionata la ROM. In questo modo si ha un grande spreco di memoria: nel programma ESEMPIO DI USR del capitolo 10 vedrai come porre in memoria una routine in linguaggio macchina, senza sprecare memoria.
Vediamo ora come funziona la routine GRAF14, con riferimento agli indirizzi esadecimali.

.3FE9: maschera gli interrupt per poter leggere continuamente il registro di linea.
.3FEA: carica nell'accumulatore i bit 7-0 del registro di linea.
.3FED: controlla se e' arrivato a $CC, non controlla il bit 8 perche' questo non puo' che essere 0 quando 7-0 contengono $CC: infatti il registro di linea puo' contenere da $00 a $138.
.3FEF: se il registro di linea non contiene ancora $CC torna a leggere.
.3FF1: carica nell'accumulatore $14 (20, che e' il codice ASCII

di DELETE).
.3FF3: lo stampa.
.3FF6/3FFB: pone a 1 i bit 0-2 del registro di scorrimento orizzontale.
.3FFE: sente nuovamente gli interrupt.
.3FFF: torna al BASIC.

Per uno scorrimento fine verticale le cose si complicano ulteriormente: infatti se chiediamo uno scorrimento fine mentre il registro di linea e' in una posizione compresa tra la linea 4 e la 203, otteniamo uno spiacevole sfarfallio dello schermo e lo stesso vale, a maggior ragione, quando chiediamo un'operazione del tipo 7 passi indietro piu' 8 avanti. Per risolvere questi problemi bisogna quindi sincronizzare gli scorrimenti fini col registro di linea e compiere lo scorrimento degli 8 passi in maniera molto rapida. Purtroppo neanche una routine in linguaggio macchina riesce ad essere cosi' veloce da far scorrere lo schermo mentre il registro di linea e' tra 204 e 3. Abbiamo quindi fatto una routine in linguaggio macchina, la GRAF16, che usa 3 mappe video e degli attributi. Nella solita coppia, da $0800 a $0BE8 e da $0C00 a $0FE8, poniamo, con l'istruzione BASIC CHAR, le nuove stringhe; le altre due coppie, che sono visualizzate alternativamente da TED, occupano le locazioni da $3000 a $33E8 e da $3400 a $37E8 la prima coppia, da $3800 a $3BE8 e da $3C00 a $3FE8 la seconda. In questo modo, quando TED mostra una coppia, vengono trasferiti i dati della mappa video e attributi normali nell'altra. Al fatidico momento degli 8 passi avanti e 7 indietro, non facciamo altro che cambiare mappa video e attributi e compiere 7 passi indietro (operazione sufficentemente veloce): potremo quindi, con calma, aggiungere i nuovi dati nelle mappe normali e trasferire il tutto nelle mappe non visualizzate. Vediamo, nel dettaglio, come funziona GRAF16:

```
MONITOR
   PC  SR AC XR YR SP
; 0000 00 00 00 00 F8
. 2FB0  78        SEI
. 2FB1  AD 1D FF  LDA $FF1D
. 2FB4  C9 CC     CMP #$CC
. 2FB6  D0 F9     BNE $2FB1
. 2FB8  AD FF 2F  LDA $2FFF
. 2FBB  D0 0D     BNE $2FCA
. 2FBD  AD 14 FF  LDA $FF14
. 2FC0  49 08     EOR #$08
. 2FC2  8D 14 FF  STA $FF14
. 2FC5  A9 08     LDA #$08
. 2FC7  8D FF 2F  STA $2FFF
. 2FCA  CE FF 2F  DEC $2FFF
. 2FCD  AD 06 FF  LDA $FF06
. 2FD0  29 F0     AND #$F0
. 2FD2  18        CLC
. 2FD3  6D FF 2F  ADC $2FFF
. 2FD6  8D 06 FF  STA $FF06
. 2FD9  60        RTS
. 2FDA  AD 14 FF  LDA $FF14
. 2FDD  29 F8     AND #$F8
. 2FDF  49 08     EOR #$08
. 2FE1  85 04     STA $04
. 2FE3  A9 08     LDA #$08
. 2FE5  85 06     STA $06
. 2FE7  A0 00     LDY #$00
. 2FE9  84 03     STY $03
. 2FEB  84 05     STY $05
. 2FED  B1 05     LDA ($05),Y
. 2FEF  91 03     STA ($03),Y
. 2FF1  C8        INY
. 2FF2  D0 F9     BNE $2FED
```

```
. 2FF4   E6 04    INC $04          . 2FFA   C9 10    CMP #$10
. 2FF6   E6 06    INC $06          . 2FFC   D0 EF    BNE $2FED
. 2FF8   A5 06    LDA $06          . 2FFE   60       RTS
```

I byte usati da questo programma sono:
.$2FFF: POSIZIONE in cui TED visualizza lo schermo (da 0 a 7).
.$03,$04: PUNTATORE 1 che indica l'indirizzo di partenza delle mappe in cui vanno trasferiti i dati.
.$05,$06: PUNTATORE 2 che indica l'indirizzo di partenza delle mappe normali.

GRAF16 e' composto da due routine: la prima fa scorrere lo schermo di una linea ogni volta che viene chiamata e, se necessario, cambia la pagina visualizzata; la seconda (che verra' chiamata dal BASIC dopo aver inserito i nuovi dati nelle mappe normali), trasferisce i dati dalle mappe normali a quelle non visualizzate.
Ecco lo schema a blocchi di GRAF16:

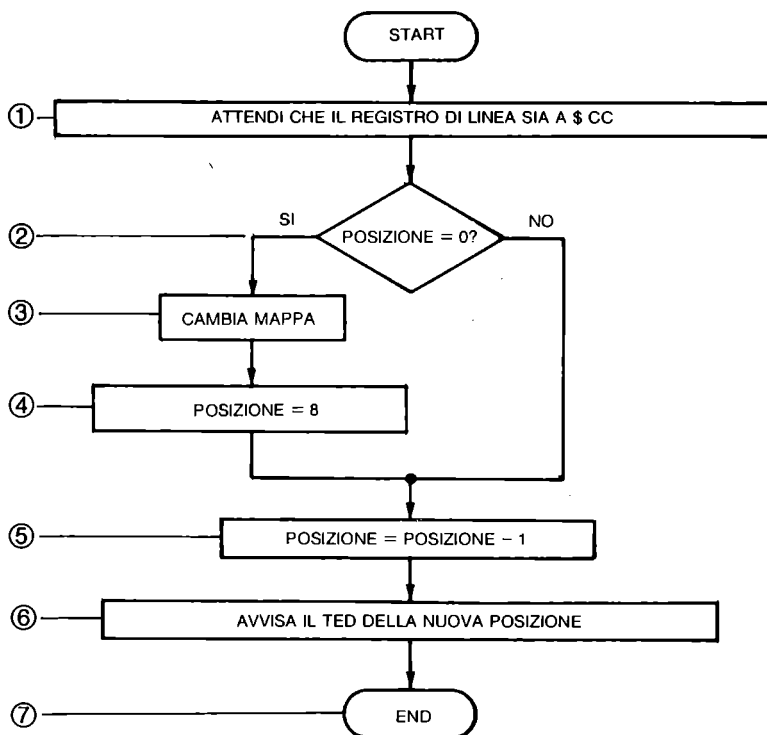
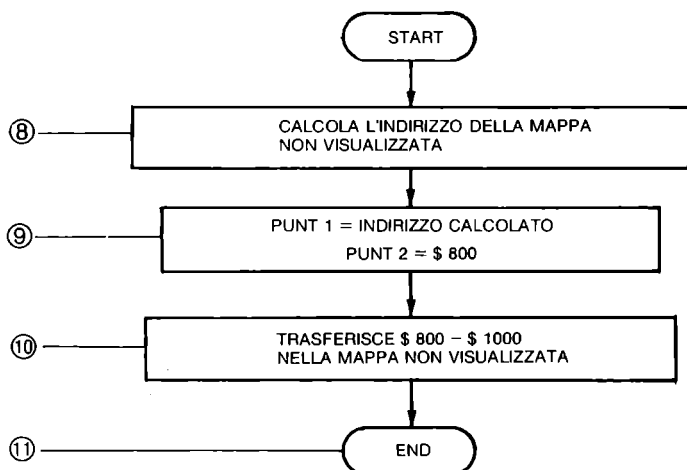



Figura 9.1 Diagramma a blocchi sottoprogramma GRAF16

Ecco infine la corrispondenza tra i blocchi dello schema e le linee del programma, riferendoci agli indirizzi esadecimali:

BLOCCO 1 : linee 2FB0-2FB6.
BLOCCO 2 : linee 2FB8-2FBB.
BLOCCO 3 : linee 2FBD-2FC2.
BLOCCO 4 : linee 2FC5-2FC7.
BLOCCO 5 : linee 2FCA-2FCD.
BLOCCO 6 : linee 2FD0-2FD6.
BLOCCO 7 : linee 2FD9.
BLOCCO 8 : linee 2FDA-2FDF.
BLOCCO 9 : linee 2FE1-2FEB.
BLOCCO 10 : linee 2FED-2FFC.
BLOCCO 11 : linee 2FFE.

Vediamo ora come il BASIC gestisce queste due routine:

```
0 REM GRAF15
10 POKE56,47:POKE55,176:CLR:TRAP500
30 FORI=0TO78:READA:POKE12208+I,A:NEXT
50 READN
60 DIMCL(N-1),LU(N-1),LN$(N-1)
70 FORI=0TON-1
80 READCL(I),LU(I),LN$(I)
110 NEXT
120 COLOR0,1:COLOR4,1:SCNCLR
```

```
130 POKE65300,48:SYS12250
140 POKE65300,56:SYS12250
170 POKE12287,7
175 DO
180 FORI=0TON-1
190 COLOR1,CL(I),LU(I):CHAR,0,24,CHR$(17)+LN$(I)
210 SYS12250
220 SYS12208
230 FORJ=1TO7
240 FORK=1TO60:NEXT
250 SYS12208
260 NEXTJ
270 NEXTI
280 LOOP
500 COLOR1,2,7:POKE65286,27:POKE65300,15
510 PRINT:PRINTERR$(ER):PRINT"IN"EL:END
1000 DATA120,173,29,255,201,204,208,249
1010 DATA173,255,47,208,13,173,20,255
1020 DATA73,8,141,20,255,169,8,141
1030 DATA255,47,206,255,47,173,6,255
1040 DATA41,240,24,109,255,47,141,6
1050 DATA255,96,173,20,255,41,248,73
1060 DATA8,133,4,169,8,133,6,160
1070 DATA0,132,3,132,5,177,5,145
1080 DATA3,200,208,249,230,4,230,6
1090 DATA165,6,201,16,208,239,96
10000 DATA10
10010 DATA3,3,"               ********************"
10020 DATA5,4,"               *                  *"
10030 DATA4,4,"               * COMMODORE PLUS-4 *"
10040 DATA6,4,"               *                  *"
10050 DATA9,5,"               * SMOOTH SCROLLING *"
10060 DATA13,4,"              *                  *"
10070 DATA8,5,"               ********************"
10080 DATA1,1,""
10090 DATA1,1,""
10100 DATA1,1,""
```

COMMENTO A GRAF15

.10: pone la fine della memoria in 12208 ($2FB0) dove inizia il programma in linguaggio macchina, in $3000 inizia la prima mappa attributi, in $3400 la prima mappa video, in $3800 inizia la seconda mappa attributi, in $3C00 la seconda mappa video. Indica che la routine di errore parte dalla linea 500.

.30: pone in memoria GRAF19.

.50: legge il numero di stringhe che dovra' visualizzare.

.60: dimensiona 3 vettori di tanti elementi quante sono le stringhe: per ogni stringa memorizza il colore, la luminosita' e la stringa stessa.

.70/110: riempie i vettori.
.120: schermo e sfondo neri, pulisce lo schermo.
.130: mappa attributi in $3000 (e quindi mappa video in $3400); salta alla routine che trasferisce nell'altra coppia il contenuto delle mappe normali (che al momento sono vuote).
.140: mappa attributi in $3800 (e quindi mappa video in $3C00); salta alla stessa routine pulendo anche l'altra coppia di mappe.
.170: pone 7 nel byte che indica la posizione del video alla routine in linguaggio macchina ($2FFF).
.175: inizializza un ciclo senza fine che si chiude in 280: per uscire dal programma basta premere RUN/STOP.
.180: per ogni stinga da visualizzare esegue da qui a 260.
.190: seleziona colore e luminosita' richiesti; stampa la stringa sulla mappa normale provocando uno scorrimento (con CHR$(17)).
.210: salta alla routine che cambia mappe e trasferisce le mappe normali nelle mappe non visualizzate.
.220: salta alla routine che provoca lo scorrimento fine.
.230/260: provoca i 7 rimanenti scorrimenti fini.
.270: prossima stringa.
.280: chiude il ciclo aperto in 175.
.500/510: abbiamo programmato una routine di errore perche' il SISTEMA OPERATIVO segnala l'errore sulla mappa normale; la routine di errore, invece, lo segnala dopo aver riportato le mappe in posizione normale e aver riportato lo schermo a 25 colonne, e arresta il programma.
.1000/1090: dati relativi alla routine GRAF19.
.10000: le 10 stringhe da visualizzare.
.10010/10100: colori, luminosita' e caratteri di ogni stringa.

Puoi ovviamente cambiare le ultime 11 linee del programma e far scorrere il messaggio che preferisci.

9.9 RIEPILOGO

In questo paragrafo abbiamo riassunto le istruzioni necessarie per sfruttare le capacita' di TED che hai visto in questo capitolo: quando ti sarai impadronito delle tecniche che ti permettono di sfruttare in pieno TED, ti bastera' consultare queste pagine per ricordarti quali sono i registri che ti interessano.

- MAPPA VIDEO E ATTRIBUTI: per cambiare l'indirizzo di partenza della mappa video e attributi devi dare le istruzioni:

```
POKE 65300,N    oppure:LDA #$N
                       STA $FF14
```

dove N*256 e' l'indirizzo del primo byte della nuova mappa attributi, la nuova mappa video sara' automaticamente posta 1024 byte oltre. N deve essere un numero divisibile per 8: se non lo e' TED considera (N/8)*8 (cioe' non considera i bit 2-0). Normalmente la mappa attributi e' posta in $0800 e quindi il numero N necessario a riportare il calcolatore nello stato normale e' 8.

DESCRIZIONE DEI CARATTERI: per cambiare l'indirizzo del primo byte di descrizione dei caratteri devi dare le istruzioni:

```
POKE 65298,PEEK(65298) AND 251
POKE 65299,(PEEK(65299) AND 3) + N
```

oppure

```
LDA $FF12
AND #$FB
STA $FF12
LDA $FF13
AND #$03
ORA #$N
STA $FF13
```

Dove N*256 e' l'indirizzo desiderato e N e' divisibile per 4 (pena il malfunzionamento del sistema). Per riportare la mappa dei caratteri in ROM devi sostituire l'operazione AND 251 (AND #$FB) con OR 4 (ORA #$04) e N vale 208 (#$D0)

CARATTERI A SFONDO PROGRAMMABILE: per entrare in modo sfondo programmabile devi dare le istruzioni:

```
POKE 65286,PEEK(65286) OR 64
POKE 65302,A
POKE 65303,B
POKE 65304,C
```

oppure

```
LDA $FF06
ORA #$40
STA $FF06
LDA #$A
STA $FF16
LDA #$B
STA $FF17
LDA #$C
STA $FF18
```

dove A, B e C indicano i colori e le luminosita' dello sfondo 1, 2 e 3 rispettivamente e sono codificati come L*16+K-1, con L luminosita' e K colore.
Per uscire dal modo sfondo programmabile devi dare le istruzioni:

```
POKE 65286,PEEK(65286) AND 191
```

oppure

```
LDA $FF06
AND #$BF
STA $FF06
```

- MODO MULTICOLORE: per entrare in modo multicolore devi dare le istruzioni:

```
POKE 65287,PEEK(65287) OR 16
COLOR1,A1,A2
COLOR3,B1,B2
POKE 65303,C1*16+C2-1
```

oppure

```
LDA $FF07
ORA #$10
STA $FF07
LDA #$B1*16+B2-1
STA $FF16
LDA #$C1*16+C2-1
STA $FF17
```

dove A1, B1 e C1 sono i codici delle luminosita' desiderate e A2, B2 e C2 i codici dei colori desiderati. Nelle routine in linguaggio macchina, devi porre il numero A1*16+A2-1 nel byte della mappa degli attributi corrispondente al carattere che vuoi colorare cosi'. Ricorda che A2 deve essere maggiore di 7 altrimenti il carattere verra' visualizzato in modo alta risoluzione.

ANNULAMENTO DELLO SCHERMO: per ottenere l'annullamento dello schermo ti basta dare le istruzioni:

```
POKE 65286,PEEK(65286) AND 239
```

oppure

```
LDA $FF06
AND #$EF
STA $FF06
```

per tornare allo schermo normale basta cambiare l'operazione AND 239 (AND #$EF) con OR 64 (ORA #$40).

CAPITOLO 10

TECNICHE DI PROGRAMMAZIONE E ESEMPI

10.1 INTRODUZIONE

In questo capitolo riportiamo alcuni esempi di programmi, discutendone l'impostazione e commentandoli.

10.2 DIVISIONE PAROLE IN SILLABE

Alcuni programmi di ELABORAZIONE TESTI (Word Processing) producono dei file di testo di tipo sequenziale ASCII, che contengono il testo, ma non provvedono automaticamente alla separazione delle parole in sillabe con il trattino fantasma, da utilizzare eventualmente in fase di stampa per andare a capo.
Abbiamo preparato il programma HYPHEN (trattino in inglese) per ottenere di leggere un file di testo, prodotto dal programma EASY SCRIPT del COMMODORE 64, e produrre un file di testo modificato, con altro nome e con le parole divise in due parti da un trattino fantasma (SHIFT-@), riconosciuto come tale nella fase di stampa da EASY SCRIPT.

Per separare in due parti le parole abbiamo usato un algoritmo empirico. Abbiamo osservato che e' possibile dividere le lettere che compongono le parole in tre categorie:

- Vocali o mute-liquide-nasali (a,e,i,o,u,h,l,m,n,r)
- Consonanti mute-liquide-nasali (h,l,m,n,r)
- Consonanti dure (b,c,d,f,g,p,q,s,t,v,z)

Abbiamo creato tre vettori di variabili numeriche che consideriamo booleane: LL, VV, CC. Ognuno di questi 3 vettori ha 255 elementi, uno per ogni codice ASCII. Il vettore VV indica quali lettere sono VOCALI o LIQUIDE: VV(65), ad esempio, viene posto a vero (-1), perche' la lettera A, che ha codice ASCII 65, e' vocale. Ugualmente il vettore LL indica quali lettere sono MUTE-LIQUIDE-NASALI, e il vettore CC indica le lettere che sono consonanti dure. In conseguenza gli elementi dei 3 vettori contengono o 0 o -1.

Per effettuare la separazione in sillabe osserviamo un gruppo di tre lettere e poniamo la divisione tra il primo e il secondo dei tre caratteri che analizziamo, a patto che si verifichi una delle seguenti condizioni:

1- Le prime due lettere sono uguali e sono consonanti (liquide-nasali o dure): e' il caso della doppia.

2- Non separiamo se la prima lettera e' S: dopo la S non si separa, se non per la doppia, gia' considerata.

3- Vocale o liquida - S - qualunque lettera esclusa S

4- Vocale o liquida - consonante dura - vocale o liquida

5- Vocale o liquida - liquida - vocale

```
1 REM HYPHEN
3 SH$="—":REM TRATTINO FANTASMA
5 REM VARIABILI DI LAVORO
7 DIMLL(255),VV(255),CC(255)
9 DIMA$(3),L(3),V(3),C(3)
11 FORI=1TO5:READA$:VV(ASC(A$))=-1:NEXTI
13 FORI=6TO10:READA$:VV(ASC(A$))=-1
15 LL(ASC(A$))=-1:NEXTI
17 FORI=11TO21:READA$:CC(ASC(A$))=-1:NEXTI
19 REM RICHIESTA NOMI FILE INPUT E OUTPUT
21 INPUT"FILE INPUT";FI$
23 INPUT"FILE OUTPUT";FO$
25 REM APERTURA FILE
27 OPEN2,8,2,FI$:OPEN3,8,3,FO$+",S,W"
29 REM CICLO DI LETTURA DI UNA PAROLA
31 A$=""
33 GET#2,B$:RS=ST
35 IFB$<>" "ANDB$<>CHR$(13)THENA$=A$+B$:GOTO47
37 GOSUB53:REM VA A METTERE TRATTINO
39 IFRS=0THENPRINT#3,B$;:PRINTB$;:GOTO31
41 REM CHIUDE SE FINITO FILE DI INPUT
43 CLOSE3:CLOSE2
45 END
47 IFRS=0THEN33
49 GOTO37
51 REM ROUTINE CHE DIVIDE CON TRATTINO
53 IFLEN(A$)<4THENPRINTA$;:PRINT#3,A$;:RETURN
55 FORI=1TOLEN(A$)-2
57 A$(1)=MID$(A$,I,1)
59 A$(2)=MID$(A$,I+1,1)
61 A$(3)=MID$(A$,I+2,1)
63 PRINTA$(1);:PRINT#3,A$(1);:GOSUB91
65 IFA$(1)=A$(2)ANDNOT(V(1)ANDNOTL(1))THEN77
67 IFA$(1)="S"THEN79
69 IFV(1)ANDA$(2)="S"ANDA$(3)<>"S"THEN77
71 IFV(1)ANDC(2)ANDV(3)THEN77
73 IFV(1)ANDL(2)AND(V(3)ANDNOTL(3))THEN77
75 GOTO79
77 IFI>LEN(A$)/2.5THEN83
79 NEXTI:PRINTA$(2);A$(3);
```

```
81 PRINT#3,A$(2)A$(3);:RETURN
83 PRINT"-";:PRINT#3,SH$;
85 PRINTMID$(A$,I+1,254);
87 PRINT#3,MID$(A$,I+1,254);:RETURN
89 REM ROUTINE CHE PRELEVA 3 CARATTERI
91 FORJ=1TO3
93 C(J)=CC(ASC(A$(J)))
95 V(J)=VV(ASC(A$(J)))
97 L(J)=LL(ASC(A$(J)))
99 NEXT:RETURN
101 REM VOCALI E CONSONANTI
103 REM VOCALI
105 DATA A,E,I,O,U
107 REM MUTE, LIQUIDE E NASALI (LIQUIDE)
109 DATA H,L,M,N,R
111 REM LABIALI, DENTALI, GUTTURALI (CONSONANTI)
113 DATA B,C,D,F,G,P,Q,S,T,V,Z
```

COMMENTO A HYPHEN

.3: La variabile SH$ contiene il carattere da porre nel file di testo nei punti dove si puo' dividere una parola. Il carattere tra virgolette e' SHIFT-@ per il programma EASY SCRIPT del

COMMODORE 64.

.5/17: Dimensiona e riempie i tre vettori booleani.

.19/27: Chiede i nomi dei files e li apre, uno in lettura, l'altro in scrittura.

.29/39 e 47/49: Riceve in A$ una parola dal file in ingresso e la passa al sottoprogramma in 53. Queste linee vengono ripetute fino alla fine del file di ingresso.

.41/45: Chiude i file e termina

.53: Se A$ ha meno di 4 caratteri non viene separata.

.55/99: Ciclo che viene eseguito una volta per ogni carattere di A$, meno che per gli ultimi due caratteri.

.65: Verifica condizione 1.

.67: Verifica condizione 2.

.69: Verifica condizione 3.

.71: Verifica condizione 4.

.73: Verifica condizione 5.

.75: Se non si e' verificata nessuna delle condizioni, la parola non viene separata con il trattino fantasma.

.77: Se e' stata superata la meta' della parola, viene aggiunto nel file SH$ (il trattino fantasma) e stampato sul video un trattino. Una parola puo' contenere un solo trattino fantasma. Puoi cambiare questa linea e porre tutti i trattini fantasma possibili nella parola, se il tuo programma di stampa non si ferma al primo, come EASY SCRIPT del COMMODORE 64.

.87: Chiude il ciclo iniziato a 55 e ritorna dal sottoprogramma alla fine del ciclo.

.91/99: Aggiorna i valori di tre vettori booleani temporanei in funzione dei tre caratteri da analizzare, ponendo, ad esempio, a C(2)=-1 se il secondo dei tre caratteri in questione e' consonante dura.

.101/113: Linee DATA che contengono le lettere. Non scambiare ne' l'ordine delle linee ne' dei caratteri.

10.3 DISEGNO DEI CARATTERI

La figura C.4 dell'appendice C riporta i caratteri ingranditi, facendo corrispondere un asterisco ad ogni puntino del carattere, cioe' ad ogni bit 1 della sua descrizione.
La figura e' stampata dal programma DISECAR, che riportiamo in questo paragrafo.
Il problema consiste nell'andare a prelevare dalla ROM la descrizione di un carattere che ha un determinato codice, espresso in D/CODE. Il COMMODORE PLUS-4 ha memorizzate in ROM le descrizioni dei 128 caratteri del set maiuscolo/grafico e dei 128 caratteri del set minuscolo/maiuscolo. Queste descrizioni occupano in tutto 2K byte (2048 byte). I caratteri in campo inverso si ottengono scambiando i bit 0 con bit 1 e viceversa. La difficolta' consiste nel fatto che non si puo' accedere dal BASIC alla ROM che contiene le descrizioni, che si trovano dal byte 53248 ($D000) al byte 55295 ($D7FF). Per questa ragione, prima di caricare in memoria il programma DISECAR, si devono trasferire dalla ROM in RAM le descrizioni dei caratteri, con il comando MONITOR e la funzione T di trasferimento. Per lavorare si deve procedere cosi':

.1) eseguire in modo immediato:

```
POKE 55,0:POKE 56,55
```

per abbassare il TOP della memoria dedicata al BASIC a 14079 (36FFH).

.2) eseguire in modo immediato:

```
MONITOR
T D000 D7FF 3700
```

che trasferisce i 2048 byte che stanno da D000 a D7FF in quelli che iniziano in 3700 (che corrisponde all'indirizzo decimale 14080). L'ultimo byte occupato risulta quello di indirizzo 16127 (3EFFH).

.3) caricare ed eseguire il programma DISECAR.

Il programma chiede se vuoi ottenere il risultato su video o stampante, poi chiede il set di caratteri che desideri e il codice del carattere in D/CODE, tra 0 e 255. Se il codice supera 127,

esso viene diminuito di 128 e viene posto a 1 lo switch SW, per ricordare che il carattere deve essere in campo inverso.
Viene poi calcolato il valore del puntatore per prelevare gli 8 byte della descrizione del carattere. Viene stampata un'intestazione e poi il carattere ingrandito usando degli asterischi, il valore binario e decimale di ogni byte della descrizione. Se SW=1 vengono invertiti i bit della descrizione per ottenere il campo inverso.
Il programma termina dopo aver stampato un risultato, per proseguire devi scrivere ancora RUN.
Ricorda dopo aver eseguito il programma di rimettere al valore usuale i byte 55 e 56, oppure di eseguire un RESET.

```
1 REM DISECAR
3 INPUT"[][]RISULTATI SU STAMPANTE? (S/N) ";R$
5 PRINT:IFR$="N"THEN17
7 REM APRE LA STAMPANTE E STAMPA TITOLO
9 OPEN4,4:PRINT#4
11 PRINT#4,"***STAMPA IMMAGINE CARATTERI***"
13 PRINT#4
15 REM SCELTA SET CARATTERI
17 D$="":PRINT"QUALE SET VUOI USARE: "
19 PRINT"  1  PER SET MAIUSCOLO/GRAFICO"
21 PRINT"  2  PER SET MAIUSCOLO/MINUSCOLO"
23 INPUT"[][]  OPZIONE:";D$:IFD$=""THEN101
25 IFD$<>"1"ANDD$<>"2"THENPRINT"[][][][]":GOTO23
27 M$=" SET MAIUSCOLO/MINUSCOLO"
29 IFD$="1"THENM$=" SET MAIUSCOLO/GRAFICO"
31 REM RICHIESTA D-CODE CARATTERE DA DISEGNARE
33 PRINT"[][]SCRIVI IL CODICE DEL CARATTERE"
35 PRINT"IN D/CODE";:SW=0:INPUTD:PRINT
37 IFD<0ORD>255THENPRINT"[][][][][][]":GOTO27
39 REM SW=1 SE CODICE >127
41 D1=D:IFD>127THEND=D-128:SW=1
43 REM PRELEVAMENTO DESCRIZIONE
45 A=14080+(VAL(D$)-1)*1024
47 FORK=0TO7:D(K)=PEEK(A+D*8+K):NEXTK
49 REM STAMPA DATI INIZIALI
51 PRINT"[][][][]D/CODE=";D;M$:PRINT
53 PRINT"CARATTERE CORRISP. AGLI 8 BYTES:"
55 PRINT
57 PRINT"CARATTERE  VAL. BINARIO  VAL. DEC."
59 PRINT:PRINT:IFR$="N"THEN73
61 PRINT#4,"D/CODE=";D1;M$:PRINT#4
63 PRINT#4,"CARATTERE CORRISP. AGLI 8 BYTES:"
65 PRINT#4
67 PRINT#4," CARATTERE   VAL. BINARIO   ";
69 REM PREPARAZIONE E STAMPA CARATTERE
71 PRINT#4,"VAL. DEC.":PRINT#4
73 FORK=0TO7:D$(K)="":E$(K)="":B=D(K)
75 FORJ=0TO7:IFINT(B/2↑(7-J))=0THEN83
```

```
77 IFSW=1THEND$(K)=D$(K)+" ":E$(K)=E$(K)+"0":GOT
081
79 D$(K)=D$(K)+"*":E$(K)=E$(K)+"1"
81 B=B-2↑(7-J):GOTO87
83 IFSW=1THEND$(K)=D$(K)+"*":E$(K)=E$(K)+"1":GOT
087
85 D$(K)=D$(K)+" ":E$(K)=E$(K)+"0"
87 NEXTJ:NEXTK:FORK=0TO7
89 PRINTD$(K);"     ";E$(K);"       ";D(K)
91 IFR$="N"THEN97
93 PRINT#4," ";D$(K);"       ";
95 PRINT#4,E$(K);"      ";D(K)
97 NEXTK
99 REM CHIUSURA STAMPANTE
101 IFR$="S"THENPRINT#4:CLOSE4
103 STOP
```

10.4 CONTATORI

Ci siamo proposti di estrarre un gruppo di numeri a caso, compresi nell'intervallo 0-14000, e di calcolare le frequenze con le quali i numeri cadono in uno dei 14 intervalli che seguono:

0- 1000	1001- 2000
2001- 3000	3001- 4000
4001- 5000	5001- 6000
6001- 7000	7001- 8000
8001- 9000	9001-10000
10001-11000	11001-12000
12001-13000	13001-14000

Il programma all'inizio chiede quanti numeri a caso vuoi estrarre e quale argomento iniziale vuoi usare per la funzione RND. Infatti dall'argomento iniziale dipende la sequenza:

.l'argomento negativo fa partire la sequenza di estrazione da un punto determinato; se usi lo stesso numero negativo ottiene sempre gli stessi numeri ogni volta che fai girare il programma;

.l'argomento nullo fa prelevare il seme per definire l'inizio delle estrazioni dal contatore TI, quindi questo dipende dal tempo trascorso dall'accensione del calcolatore;

.l'argomento positivo fa proseguire l'estrazione dalla sequenza gia' in corso.

Il programma usa il seme da te fornito per eseguire un'estrazione iniziale fuori ciclo, poi prosegue con argomento positivo. Se dai un seme positivo e usi il programma subito dopo l'accensione ottieni sempre gli stessi risultati, infatti il seme iniziale e' sempre uguale. I numeri a caso estratti sono compresi tra 0 e 1; viene eseguito un semplice calcolo per ridurli

nell'intervallo 0-14000. Abbiamo scelto 14000 come limite, e 14 intervalli di 1000 valori, per poter visualizzare in un solo quadro video i risultati.
Il programma conta in un contatore generale i numeri estratti e calcola in tempo reale le frequenze andando ad aggiornare i 14 contatori.
Il quadro video presenta in 14 caselle i limiti dell'intervallo, evidenziandoli in campo inverso, e sotto ad ogni casella il valore della relativa frequenza.
Puo' presentare un certo interesse vedere come andiamo a scrivere in posizioni predeterminate del video servendoci dell'istruzione CHAR in modo testo. Nel programma abbiamo introdotto in linee DATA le "coordinate" di riga e colonna dove vogliamo scrivere; le memorizziamo in due vettori e usiamo la coppia di coordinate che ci vuole per ogni contatore nell'istruzione CHAR.
Ti facciamo notare che quando usi l'istruzione CHAR in modo testo non risulta valido il parametro "modo" (l'ultimo) che, invece, in modo grafico e' attivo e serve per ottenere il campo diretto o inverso. Noi per poter stampare in campo inverso dobbiamo usare i relativi caratteri di controllo nella stringa di stampa.
Per poterti spostare sul video puoi ricorrere alla CHAR, oppure usare i caratteri di controllo del cursore e le funzioni TAB e SPC.

```
1 REM CONTATORI
3 S$="                  "
5 I$="▯":D$="▀"
7 REM DESCRIZIONE CONTATORI
9 DATA "     0-- 1000"," 1001-- 2000"
11 DATA " 2001-- 3000"," 3001-- 4000"
13 DATA " 4001-- 5000"," 5001-- 6000"
15 DATA " 6001-- 7000"," 7001-- 8000"
17 DATA " 8001-- 9000"," 9001--10000"
19 DATA "10001--11000","11001--12000"
21 DATA "12001--13000","13001--14000"
23 REM LIMITI RIPARTIZIONE
25 DATA 13001,12001,11001,10001,9001,8001,7001
27 DATA 6001,5001,4001,3001,2001,1001
29 REM POSIZIONI STAMPA
31 DATA 2,1,2,21,5,1,5,21,8,1,8,21,11,1,11,21
33 DATA 14,1,14,21,17,1,17,21,20,1,20,21
35 DATA 3,1,3,21,6,1,6,21,9,1,9,21,12,1,12,21
37 DATA 15,1,15,21,18,1,18,21,21,1,21,21
39 REM CARICAMENTO MATRICI
41 DIMX(14),Y(14),Z(14),W(14)
43 RESTORE31
45 FORK=1TO14:READX(K),Y(K):NEXTK
47 RESTORE35
49 FORK=1TO14:READZ(K),W(K):NEXTK
51 REM INIZIALIZZAZIONE CONTATORI
```

```
53 DIMC(14):C=0
55 FORK=1TO14:C(K)=0:NEXTK
57 PRINT"SCRIVI QUANTI NUMERI A CASO VUOI"
59 PRINT"ESTRARRE TRA 0 E 14000"
61 INPUT"QUANTI: ";R
63 PRINT"ESTRAZIONE DI ";R;" NUMERI A CASO"
65 INPUT"SEME PER RND: ";S
67 PRINT"SEME PER RND: ";S
69 PRINT"PREMI UN TASTO PER CONTINUARE"
71 GETKEYA$
73 REM PREPARAZIONE QUADRO
75 RESTORE9:PRINT""
77 FORK=1TO14:READA$
79 X=X(K):Y=Y(K):Z=Z(K):W=W(K)
81 CHAR,Y,X,I$+A$+D$
83 C$=S$+STR$(C(K))
85 CHAR,W,Z,RIGHT$(C$,12)
87 NEXTK
89 CHAR,2,24,"CONTATORE GENERALE: "
91 REM ESTRAZIONE NUMERO
93 N=RND(S)
95 FORJ=1TOR:N=RND(1)
97 N=1+INT(N*14000)
99 REM CERCA INTERVALLO
101 RESTORE25
103 FORK=14TO2STEP-1
105 READA
107 IFN>=ATHENC(K)=C(K)+1:X=Z(K):Y=W(K):GOTO117
109 NEXTK
111 C(1)=C(1)+1:X=Z(1):Y=W(1):K=1:GOTO117
113 NEXTJ
115 GOTO115
117 REM STAMPA CONTATORE
119 C$=S$+STR$(C(K))
121 C=C+1
123 CHAR,Y,X,RIGHT$(C$,12)
125 C$=S$+STR$(C)
127 CHAR,21,24,RIGHT$(C$,12)
129 GOTO113
```

COMMENTO A CONTATORI

.3/5: definizione S$ con 12 spazi, I$ con il carattere di controllo RVS ON, D$ con il carattere di controllo RVS OFF.

.7/21: linee DATA con le descrizioni dei 14 contatori.

.23/27: linee DATA con i limiti per il controllo delle frequenze; tali limiti sono solo 13 e in ordine decrescente, infatti i controlli si fanno in cascata partendo dal valore piu' alto.

.29/37: linee DATA con i valori delle coordinate per le posizioni video delle descrizioni dei contatori e dei contatori.

.39/49: caricamento dei dati nelle relative matrici.
.51/55: inizializzazione contatori delle frequenze.
.57/71: richiesta dati iniziali.
.73/89: preparazione quadro video iniziale.
.91/113: ciclo di estrazione dei numeri e aggiornamento contatori.
.115: ferma il programma in modo che anche premendo un tasto non si sciupa il quadro; per uscire premi STOP.
.117/129: stampa del contatore che e' stato aggiornato.

10.5 GRAFICO TRIDIMENSIONALE

Il programma 3DP, nonostante sia scritto completamente in BASIC e sia breve, e' un programma abbastanza complesso e da' risultati molto spettacolari; esso infatti disegna funzioni reali di due variabili reali, disegna cioe' funzioni tridimensionali.
Il programma disegna funzioni definite sulla parte di piano -2<X<2, -2<Y<2 che ammettano valori tra -1 e 1. Se vuoi vedere l'andamento di una funzione su un dominio piu' ampio, ti conviene modificare la linea 500, in cui e' definita la funzione. Ad esempio, se vuoi vedere la funzione f(X,Y) su un dominio -10<X<10, -10<Y<30, ti bastera' impostare la funzione f(X*5,Y*10+10)/M dove M e' il massimo valore che ABS(f(X*5,Y*10+10)) assume nell'intervallo -2<X<2, -2<Y<2 o (che e' la stessa cosa) che la funzione f(X,Y) assume nell'intervallo -10<X<10, -10<Y<30. Il programma disegna la funzione partendo dall'osservatore e procedendo verso "l'interno del monitor"; per sapere se il punto che ha disegnato e' nascosto dal disegno precedentemente fatto, controlla se il punto che deve disegnare sullo schermo non e' compreso tra il punto piu' alto e il piu' basso di una stessa colonna di punti. Per ottenere questo avremo bisogno di due vettori di 320 elementi ciascuno (1 per colonna di punti) in cui potremo mettere rispettivamente il punto piu' alto e il punto piu' basso disegnato su una colonna. Prima di presentare il listato del programma diamo ancora un'informazione che puo' aiutare a comprenderlo: il COMMODORE PLUS-4 calcola il valore della funzione tridimensionale dato un punto del piano: tale valore viene posto nella variabile Z3 e le coordinate del punto sono X3 e Y3. X3, Y3 e Z3 sono le coordinate di un punto nello spazio. Il calcolatore dovra' ora disegnare un punto su un piano (il video) in modo che dia l'impressione della profondita'. Per far cio' usiamo l'assonometria, calcoliamo cioe' la coordinata verticale del piano (Y2) come Z3+X3/2 e la coordinata orizzontale del piano come (X2) come Y2+X3/2 (vedi Figura 10.1).

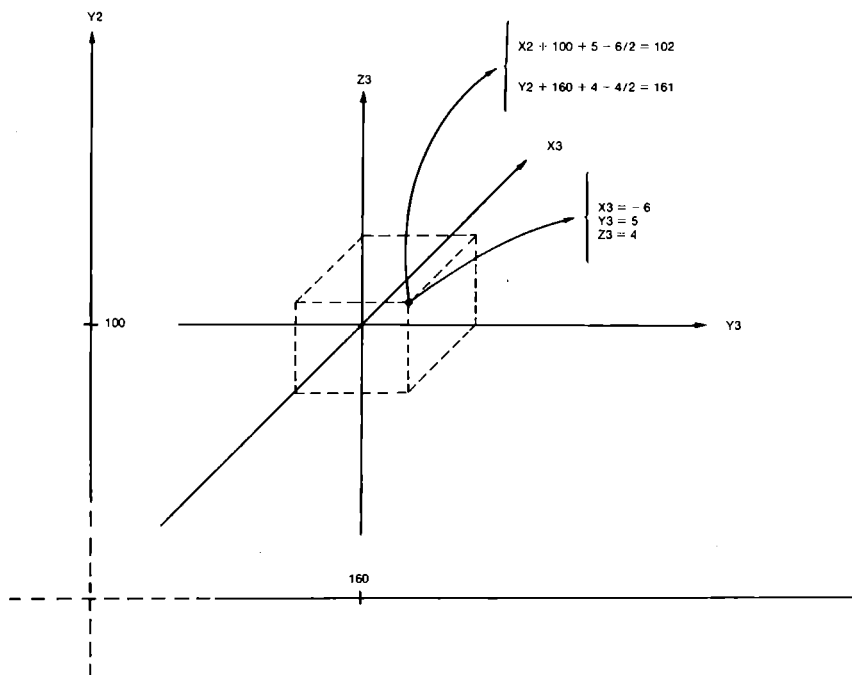



Figura 10.1 Rappresentazione in assonometria

```
0 REM 3DP
10 DIM MN%(319), MX%(319)
20 FORI=0TO319:MN%(I)=199:MX%(I)=0:NEXT I
30 COLOR0,1:COLOR1,2,7:COLOR4,1
40 GRAPHIC1,1
50 X3=-2:FORY3=-2TO2STEP.02:GOSUB500:NEXT
60 SP=.3
70 FORX3=-2TO2STEP.02:Y3=-2
80 GOSUB500:Y3=2:GOSUB500
90 RX=X3-INT(X3/SP)*SP
100 FORY3=-2+RXTO2STEPSP
110 GOSUB500
120 NEXTY3
130 FORY3=2-RXTO-2STEP-SP
140 GOSUB500
150 NEXTY3:NEXTX3
160 X3=2:FORY3=-2TO2STEP.02:GOSUB500:NEXT
170 GETKEYA$
180 GRAPHIC0:END
500 Z3=SIN(X3*Y3)
```

```
510 X2=160+(Y3+X3/2)*49:Y2=100+(Z3+X3/2)*49
520 IFY2>MX%(X2)THENMX%(X2)=Y2:GOTO550
530 IFY2<MN%(X2)THENMN%(X2)=Y2:GOTO560
540 RETURN
550 IFY2<MN%(X2)THENMN%(X2)=Y2
560 DRAW,X2,199-Y2:RETURN
```

COMMENTO A 3DP

.10: dimensiona i vettori MN e MX che conterranno i minimi e i massimi di ogni colonna.

.20: pone a 0 il vettore dei massimi e a 199 quello dei minimi.

.30: schermo e bordo neri, scritte bianche.

.40: entra in modo grafico.

.50: calcola e disegna in assonometria (appoggiandosi alla routine in 500) il valore della funzione sul segmento X=-2, -2<Y<2, cioe' il segmento piu' vicino all'osservatore.

.60: pone il passo del reticolo a 0,3. Aumentando il valore della variabile SP si otterra' un reticolo a maglie piu' larghe, diminuendolo piu' strette.

.70/80: inizializza un ciclo che incrementa la X (l'asse che va dall'osservatore verso lo schermo). Quindi calcola il valore della funzione per Y=-2 e Y=2, cioe' ai lati del dominio.

.90/100: inizializza un ciclo che incrementa la Y con passo SP, partendo da un valore tale per cui il reticolo sara' diagonale rispetto agli assi.

.110: calcola e disegna la funzione.

.120: chiude il ciclo della Y.

.130/140: esegue lo stesso lavoro per i valori della Y simmetrici, per disegnare le diagonali perpendicolari alle prime.

.150: chiude i cicli della Y e della X.

.160: come la linea 50 solo che il segmento e' X=2, -2<Y<2, cioe' il segmento piu' lontano dall'osservatore.

.170: attende la pressione di un tasto.

.180: il programma salta a questa linea anche se e' stato commesso un errore o se e' stato premuto il tasto STOP. Torna al modo testo e ferma il programma.

.500: calcola il valore della funzione tridimensionale.

.510: calcola i valori di X2 e Y2.

.520: se il valore massimo della Y su quella colonna e' minore del valore appena calcolato pone il valore massimo della colonna al valore appena calcolato e salta a 550.

.530: se il valore minimo della Y su quella colonna e' maggiore del valore appena calcolato pone il valore minimo della colonna al valore appena calcolato e salta a 560.

.540: se il valore della Y2 cade tra massimo e minimo, il punto e' nascosto dalla funzione precedentemente disegnata e quindi non

deve essere disegnato.
.550: il programma passa di qui se il valore della Y e' maggiore del valore massimo su quella colonna. Supera l'IF se e' anche piu' piccolo del valore minimo, cioe' se e' il primo punto di quella colonna. In tal caso pone anche il minimo uguale al valore attuale (il massimo era gia' stato sistemato nella linea 520).
.560: disegna il punto e torna.

Segue nella Figura 10.2 il risultato del programma 3DP, ottenuto con il programma VIDEOGRAF.

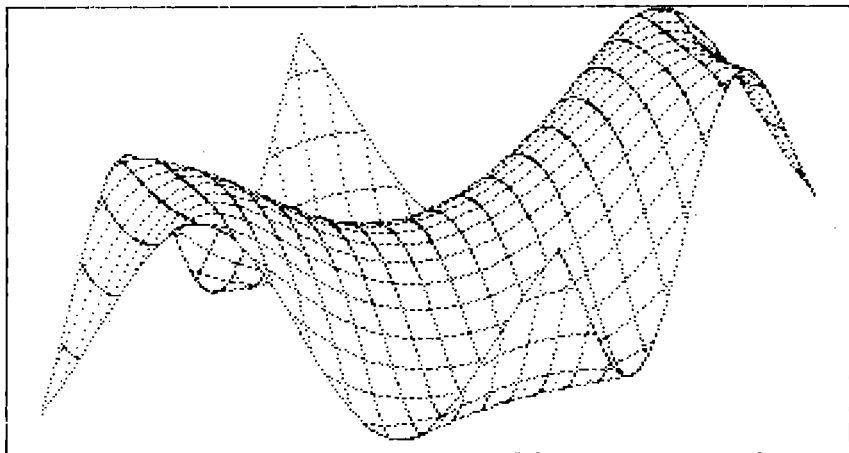



Figura 10.2 Grafico della funzione Z3=SIN(X3*Y3)

Prova a sostituire alla linea 500 le seguenti linee:

Z3=SIN(Y3*2)

Z3=SIN(X3*X3+Y3*Y3)

Z3=(X3*X3+Y3*Y3)*(X3*X3+Y3*Y3)/32-1

Nelle Figure 10.3, 10.4 e 10.5 sono riportati i risultati ottenuti modificando la linea 500.

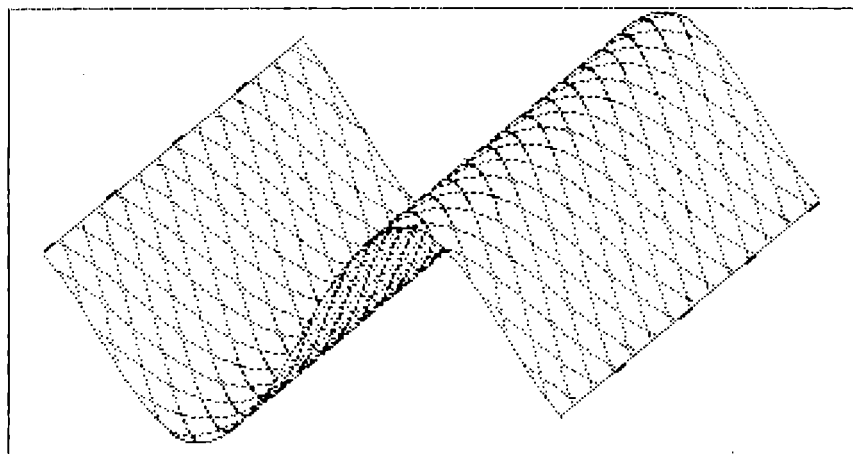



Figura 10.3 Grafico della funzione Z3=SIN(Y3*2)

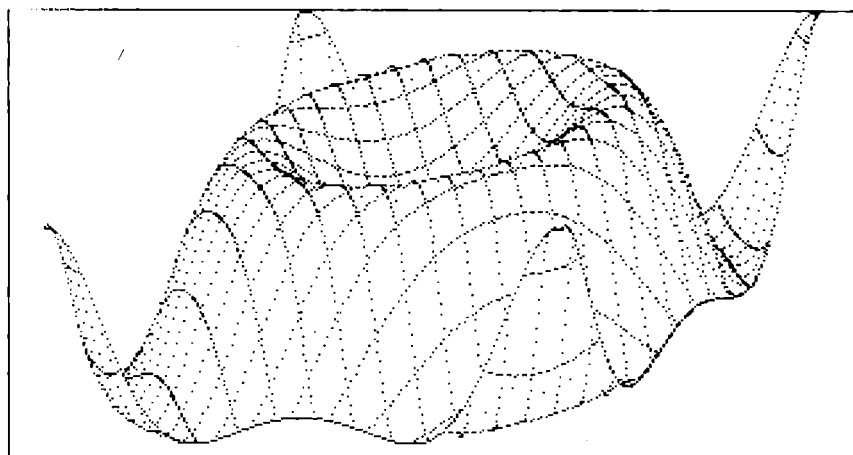



Figura 10.4 Grafico della funzione Z3=SIN(X3*X3+Y3*Y3)

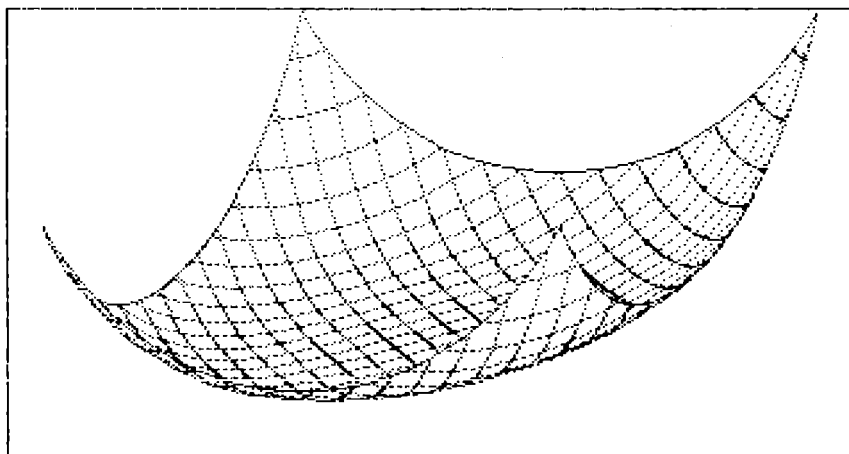



Figura 10.5 Grafico della funzione
Z3=(X3*X3+Y3*Y3)*(X3*X3+Y3*Y3)/32-1

10.6 UTILIZZO INTERRUPT

Il programma INTERRUPT mostra come sia possibile, modificando opportunamente la routine di interrupt, fare svolgere al COMMODORE PLUS-4 una funzione particolare "mentre" svolge una qualunque attivita' normale (EDITOR o esecuzione di un programma). Questo programma, infatti, fa muovere un piccolo aeroplano sulla scritta "INTERRUPT MODIFICATO" che viene posta sulla prima linea dello schermo, mentre il calcolatore e' pronto a caricare e eseguire programmi, fare calcoli e tutto cio' che puo' fare normalmente. Per realizzare il nostro scopo "intercettiamo" la routine di interrupt, facciamo le operazioni necessarie per muovere l'aeroplano e torniamo alla normale routine di interrupt. Per intercettare la routine di interrupt basta porre nei byte $0312, $0313 l'indirizzo di partenza della routine che si vuole inserire e finire la stessa con l'istruzione JMP $CE42. Questo programma mostra anche come puoi far muovere di un punto alla volta un carattere che si sovrappone ai caratteri che incontra e che li lascia inalterati una volta che li ha superati. Il nostro aeroplanino e' lungo 8 punti: per descriverlo usiamo pero' 16 byte di memoria per poterlo far scorrere di un punto alla volta (vedi Figura 10.6).
Per ricordare quali sono i due caratteri che sono coperti dalla nostra figura usiamo 2 buffer, memorizziamo cioe' nel buffer di

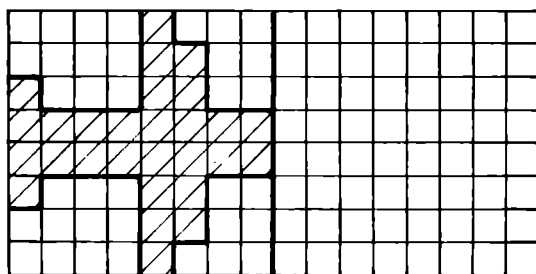
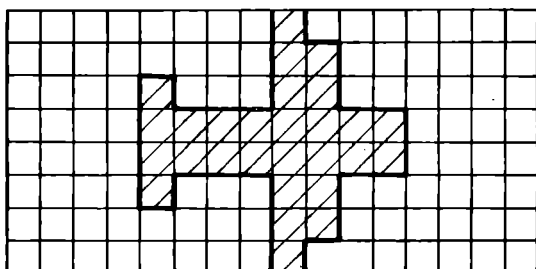
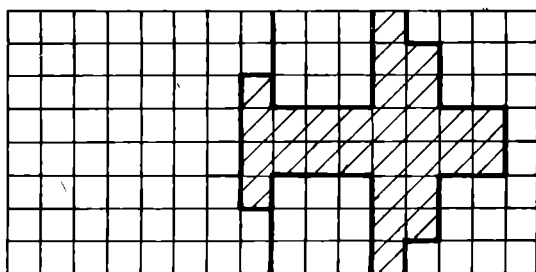



Figura 10.6 Tre posizioni dell'aeroplanino

"destra" il carattere "sotto" la punta e nel buffer di "sinistra" quello sotto la coda; fatti otto passi avanti riportiamo la figura in posizione 0 e facciamo avanzare i due caratteri di una posizione. A questo punto dovremo mettere dove prima c'era la coda il contenuto del buffer di sinistra, nel buffer di sinistra il contenuto del buffer di destra e nel buffer di destra cio' che era davanti alla figura. Infine, per poter sovrapporre l'immagine dell'aeroplanino alle immagini dei caratteri che esso "sorvola" teniamo 16 byte in cui facciamo scorrere l'immagine dell'aeroplano. Per visualizzarlo poniamo nei byte riservati ai

caratteri di D/CODE 126 e 127 l'immagine dell'aeroplano dopo aver eseguito la OR con i corrispondenti byte delle descrizioni dei caratteri i cui D/CODE sono contenti nel buffer.
Il programma BASIC INTERRUPT abbassa i puntatori di fine memoria, pone in memoria il programma in linguaggio macchina, pone la descrizione dei caratteri in RAM (da $3800 in poi; verranno pero' usati solo i caratteri minuscoli che iniziano in $3C00), seleziona il set minuscolo, scrive il messaggio, crea una finestra video, lancia la routine in linguaggio macchina e si autocancella.
Il programma in linguaggio macchina puo' essere diviso in due parti:

- la prima, INT1, (da $3B10 a $3B54) inizializza il programma.
- la seconda, INT2, (da $3B58 a $3BF9) e' la parte che verra' eseguita ad ogni chiamata di interrupt.

I byte che servono al programma in linguaggio macchina per lavorare sono:

$3B00-$3B0F: contengono la descrizione dell'aeroplano che cambia ad ogni chiamata di interrupt (vedi Figura 10.6).
$3B55: buffer di sinistra.
$3B56: buffer di destra.
$3B57: posizione dell'aeroplanino nella matrice.
$3FF0-$3FFF: sono i 16 byte che descrivono i caratteri di D/CODE 126 e 127: contengono la descrizione dell'aeroplanino dopo che ha subito la OR con le immagini dei caratteri il cui D/CODE e' nel buffer.

Ecco il listato del programma BASIC INTERRUPT:

```
0 rem interrupt
10 poke56,59:poke55,00:clr
11 fori=0to249:reada:poke15104+i,a:next
20 poke65299,(peek(65299)and3)+56
30 poke65298,peek(65298)and251
40 a$="              INTERRUPT MODIFICATO"
41 printchr$(147)chr$(14)chr$(8)a$
50 print:printchr$(27)"t":sys15120:new
1000 data8,12,204,255,255,204,12,8
1010 data0,0,0,0,0,0,0,0
1020 data120,169,212,133,4,169,60,133
1030 data6,160,0,132,3,132,5,177
1040 data3,145,5,200,208,249,230,4
1050 data230,6,165;4,201,216,208,239
1060 data169,0,141,87,59,169,12,133
1070 data4,169,0,133,3,173,0,12
1080 data141,85,59,173,1,12,141,86
1090 data59,169,59,141,19,3,169,88
1100 data141,18,3,88,96,0,0,0
1110 data173,87,59,201,8,240,87,238
1120 data87,59,162,7,94,0,59,126
```

```
1130 data8,59,202,16,247,160,7,162
1140 data1,189,85,59,133,5,169,0
1150 data133,6,6,5,38,6,6,5
1160 data38,6,6,5,38,6,169,60
1170 data24,101,6,133,6,177,5,202
1180 data208,9,25,8,59,153,248,63
1190 data76,161,59,25,0,59,153,240
1200 data63,232,202,16,204,136,16,199
1210 data160,0,169,126,145,3,200,169
1220 data127,145,3,76,66,206,169,0
1230 data141,87,59,162,7,189,8,59
1240 data157,0,59,169,0,157,8,59
1250 data202,16,242,173,85,59,160,0
1260 data145,3,173,86,59,141,85,59
1270 data230,3,165,3,201,39,208,16
1280 data169,0,133,3,173,85,59,141
1290 data39,12,173,0,12,141,85,59
1300 data160,1,177,3,141,86,59,76
1310 data109,59
```

Segue il listato del programma in linguaggio macchina, formato dalle 2 routine INT1 e INT2:

```
MONITOR
   PC  SR AC XR YR SP
; 0000 00 00 00 00 F8
. 3B10  78        SEI
. 3B11  A9 D4     LDA #$D4
. 3B13  85 04     STA $04
. 3B15  A9 3C     LDA #$3C
. 3B17  85 06     STA $06
. 3B19  A0 00     LDY #$00
. 3B1B  84 03     STY $03
. 3B1D  84 05     STY $05
. 3B1F  B1 03     LDA ($03),Y
. 3B21  91 05     STA ($05),Y
. 3B23  C8        INY
. 3B24  D0 F9     BNE $3B1F
. 3B26  E6 04     INC $04
. 3B28  E6 06     INC $06
. 3B2A  A5 04     LDA $04
. 3B2C  C9 D8     CMP #$D8
. 3B2E  D0 EF     BNE $3B1F
. 3B30  A9 00     LDA #$00
. 3B32  8D 57 3B  STA $3B57
. 3B35  A9 0C     LDA #$0C
. 3B37  85 04     STA $04
. 3B39  A9 00     LDA #$00
. 3B3B  85 03     STA $03
. 3B3D  AD 00 0C  LDA $0C00
. 3B40  8D 55 3B  STA $3B55
. 3B43  AD 01 0C  LDA $0C01
. 3B46  8D 56 3B  STA $3B56
. 3B49  A9 3B     LDA #$3B
. 3B4B  8D 13 03  STA $0313
. 3B4E  A9 58     LDA #$58
. 3B50  8D 12 03  STA $0312
. 3B53  58        CLI
. 3B54  60        RTS

MONITOR
   PC  SR AC XR YR SP
; 0000 00 00 00 00 F8
. 3B58  AD 57 3B  LDA $3B57
. 3B5B  C9 08     CMP #$08
. 3B5D  F0 57     BEQ $3BB6
. 3B5F  EE 57 3B  INC $3B57
. 3B62  A2 07     LDX #$07
. 3B64  5E 00 3B  LSR $3B00,X
. 3B67  7E 08 3B  ROR $3B08,X
. 3B6A  CA        DEX
. 3B6B  10 F7     BPL $3B64
. 3B6D  A0 07     LDY #$07
. 3B6F  A2 01     LDX #$01
. 3B71  BD 55 3B  LDA $3B55,X
. 3B74  85 05     STA $05
. 3B76  A9 00     LDA #$00
. 3B78  85 06     STA $06
. 3B7A  06 05     ASL $05
```

```
. 3B7C   26 06      ROL $06
. 3B7E   06 05      ASL $05
. 3B80   26 06      ROL $06
. 3B82   06 05      ASL $05
. 3B84   26 06      ROL $06
. 3B86   A9 3C      LDA #$3C
. 3B88   18         CLC
. 3B89   65 06      ADC $06
. 3B8B   85 06      STA $06
. 3B8D   B1 05      LDA ($05),Y
. 3B8F   CA         DEX
. 3B90   D0 09      BNE $3B9B
. 3B92   19 08 3B   ORA $3B08,Y
. 3B95   99 F8 3F   STA $3FF8,Y
. 3B98   4C A1 3B   JMP $3BA1
. 3B9B   19 00 3B   ORA $3B00,Y
. 3B9E   99 F0 3F   STA $3FF0,Y
. 3BA1   E8         INX
. 3BA2   CA         DEX
. 3BA3   10 CC      BPL $3B71
. 3BA5   88         DEY
. 3BA6   10 C7      BPL $3B6F
. 3BA8   A0 00      LDY #$00
. 3BAA   A9 7E      LDA #$7E
. 3BAC   91 03      STA ($03),Y
. 3BAE   C8         INY
. 3BAF   A9 7F      LDA #$7F
. 3BB1   91 03      STA ($03),Y
. 3BB3   4C 42 CE   JMP $CE42
. 3BB6   A9 00      LDA #$00
. 3BB8   8D 57 3B   STA $3B57
. 3BBB   A2 07      LDX #$07
. 3BBD   BD 08 3B   LDA $3B08,X
. 3BC0   9D 00 3B   STA $3B00,X
. 3BC3   A9 00      LDA #$00
. 3BC5   9D 08 3B   STA $3B08,X
. 3BC8   CA         DEX
. 3BC9   10 F2      BPL $3BBD
. 3BCB   AD 55 3B   LDA $3B55
. 3BCE   A0 00      LDY #$00
. 3BD0   91 03      STA ($03),Y
. 3BD2   AD 56 3B   LDA $3B56
. 3BD5   8D 55 3B   STA $3B55
. 3BD8   E6 03      INC $03
. 3BDA   A5 03      LDA $03
. 3BDC   C9 27      CMP #$27
. 3BDE   D0 10      BNE $3BF0
. 3BE0   A9 00      LDA #$00
. 3BE2   85 03      STA $03
. 3BE4   AD 55 3B   LDA $3B55
. 3BE7   8D 27 0C   STA $0C27
. 3BEA   AD 00 0C   LDA $0C00
. 3BED   8D 55 3B   STA $3B55
. 3BF0   A0 01      LDY #$01
. 3BF2   B1 03      LDA ($03),Y
. 3BF4   8D 56 3B   STA $3B56
. 3BF7   4C 6D 3B   JMP $3B6D
```

Nelle Figure 10.7 e 10.8 sono riportati i diagrammi a blocchi di INT1 e di INT2.

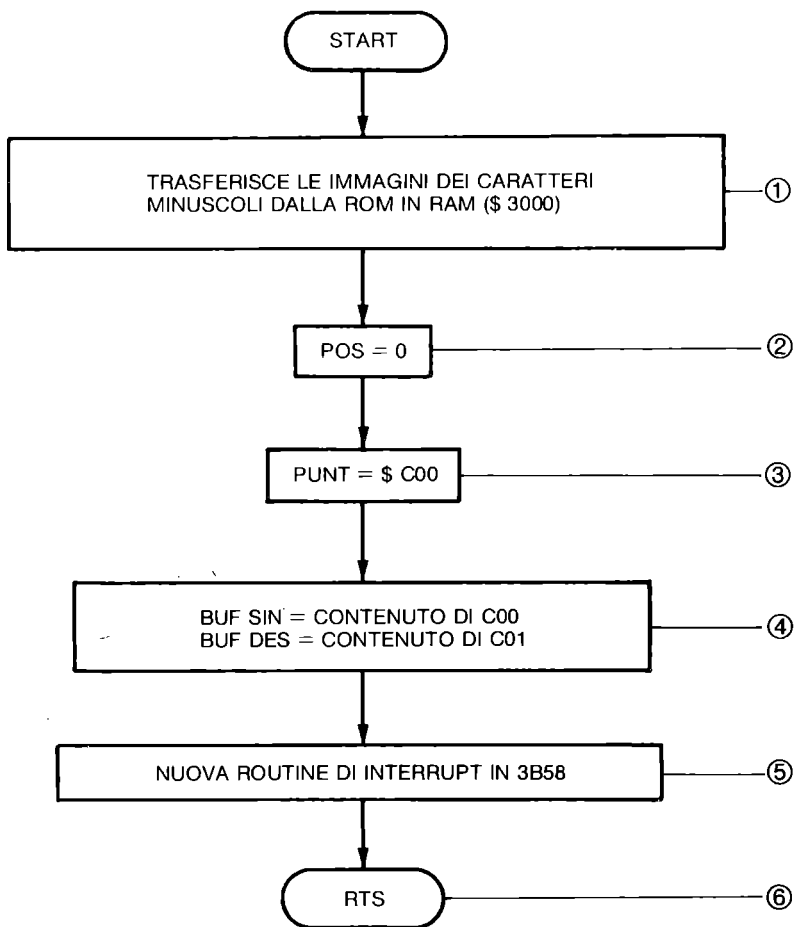



Figura 10.7 Diagramma a blocchi di INT1

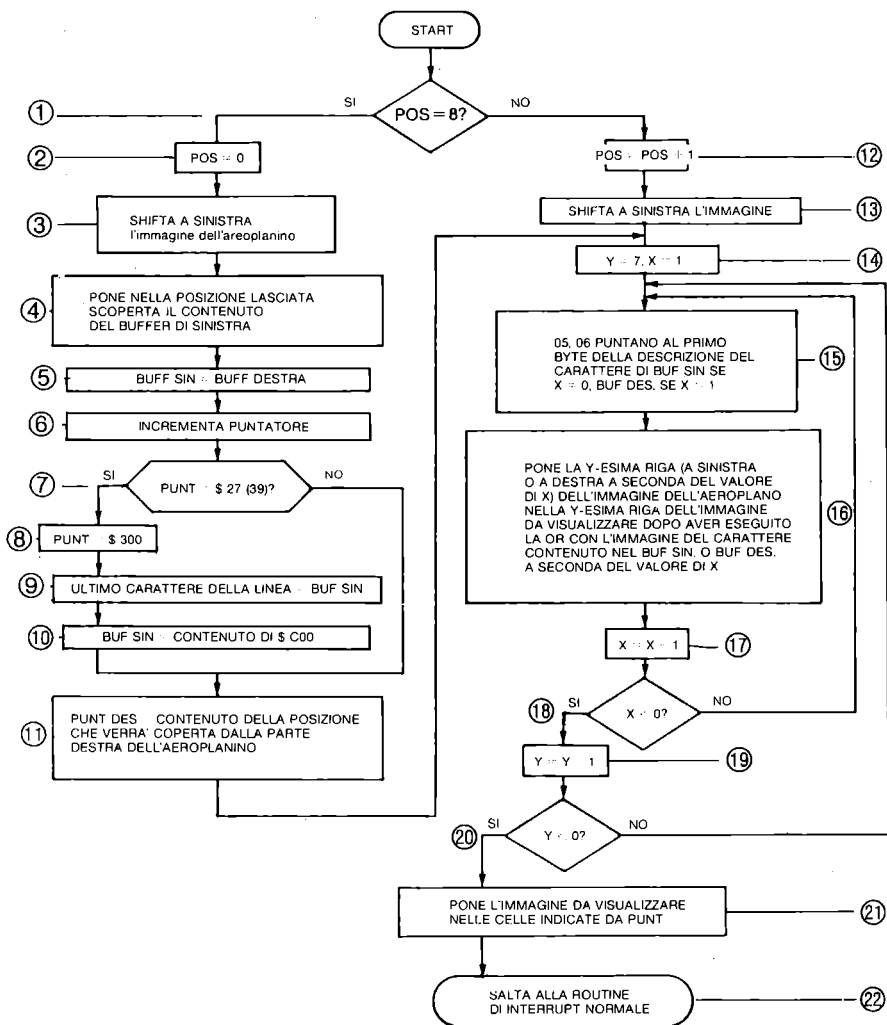



Figura 10.8 Diagramma a blocchi di INT2

Ed ecco, infine, la corrispondenza tra i blocchi dei diagrammi e il listato dei programmi in linguaggio macchina:

SCHEMA INT1:
BLOCCO 1 : linee 3B10-3B2E.
BLOCCO 2 : linee 3B30-3B32.
BLOCCO 3 : linee 3B35-3B3B.
BLOCCO 4 : linee 3B3D-3B46.
BLOCCO 5 : linee 3B49-3B50.
BLOCCO 6 : linee 3B53-3B54.

SCHEMA INT2:
BLOCCO 1 : linee 3B58-3B5D.
BLOCCO 2 : linee 3BB6-3BB8.
BLOCCO 3 : linee 3BBB-3BC9.
BLOCCO 4 : linee 3BCB-3BD0.
BLOCCO 5 : linee 3BD2-3BD5.
BLOCCO 6 : linee 3BD8.
BLOCCO 7 : linee 3BDA-3BDE.
BLOCCO 8 : linee 3BE0-3BE2.
BLOCCO 9 : linee 3BE4-3BE10.
BLOCCO 10 : linee 3BEA-3BED.
BLOCCO 11 : linee 3BF0-3BF7.
BLOCCO 12 : linee 3B5F.
BLOCCO 13 : linee 3B62-3B6B.
BLOCCO 14 : linee 3B6D-3B6F.
BLOCCO 15 : linee 3B71-3B8B.
BLOCCO 16 : linee 3B8D-3BA1.
BLOCCO 17 : linee 3BA2.
BLOCCO 18 : linee 3BA3.
BLOCCO 19 : linee 3BA5.
BLOCCO 20 : linee 3BA6.
BLOCCO 21 : linee 3BA8-3BB1.
BLOCCO 22 : linee 3BB3.

10.7 ESEMPIO DI USR

Questa routine in linguaggio macchina viene chiamata dal BASIC con la funzione USR; il parametro che viene passato e' un elemento di un vettore di stringhe. La routine, chiamata ad esempio con l'istruzione A=USR(WR$(5)), ordina il vettore WR$ a partire dall'elemento 5 in poi e pone 0 nella variabile A. Quando passi, tramite USR, l'elemento di un vettore il BASIC pone:

.In $60 $61 (96, 97) l'indirizzo del primo byte dell'intestazione del vettore (vedi Appendice E).
.In $64 $65 (100, 101) l'indirizzo del primo dei tre byte di descrizione della stringa passata come parametro.

Poiche' dopo una SYS o una USR la CPU legge la ROM da $8000 (32768) in poi, il programma deve essere posto tra $1001 e $8000. Una scelta puo' essere quella di mettere il programma poco prima di $8000 e riservare al BASIC la memoria da $1001 all'inizio della routine: questa soluzione e' consigliabile quando si conosce a priori l'occupazione del programma e delle variabili BASIC. Se pero', come in questo caso, la routine puo' essere usata da diversi programmi BASIC la cui occupazione di memoria e' a priori sconosciuta, e' preferibile porre la routine stessa a partire da $1001 e spostare l'inizio del programma BASIC alla fine della routine linguaggio macchina. Per far cio' devi inserire la linea:

```
10 POKE 1281,34: POKE 1282,16: POKE 44,18:RUN
```

Quando il calcolatore esegue questa linea pone il vettore di USR a $1022 e lancia l'esecuzione del programma BASIC in $1201 (il 18 messo nel byte di indirizzo 44 corrispone al numero esadecimale 12, piu' significativo di $1201; 34 e 16 vorrispondono a $22 e $10 rispettivamente meno e piu' significativi di $1022).
A questo punto dovrai dare i seguenti ordini in modo diretto:

```
POKE 44,18

POKE 18*256,0

NEW
```

Sposti cioe' l'inizio del BASIC a $1201, poni 0 nella cella $1200 e inizializzi il BASIC nella sua nuova area. Se non poni 0 in $1200 (18*256 decimale), il calcolatore risponde SYNTAX ERROR quando viene eseguito il comando NEW.
Puoi ora inserire il programma ESEMPIO DI USR:

```
0 REM ESEMPIO DI USR
10 FORI=0TO396:READA:POKE4130+I,A:NEXT:NEW
1000 DATA120,141,63,255,32,46,16,141
1010 DATA62,255,88,96,160,2,177,95
1020 DATA24,101,95,133,208,200,177,95
1030 DATA101,96,133,209,165,100,133,211
1040 DATA165,101,133,212,165,211,24,105
1050 DATA3,133,211,165,212,105,0,133
1060 DATA212,197,209,144,13,208,6,165
1070 DATA211,197,208,144,5,169,0,133
1080 DATA13,96,160,0,177,211,133,215
1090 DATA200,177,211,133,216,200,177,211
1100 DATA133,217,165,211,56,233,3,133
1110 DATA211,165,212,233,0,133,212,160
1120 DATA0,177,211,133,218,200,177,211
```

```
1130 DATA133,219,200,177,211,133,220,165
1140 DATA211,24,105,3,133,211,165,212
1150 DATA105,0,133,212,32,143,17,176
1160 DATA163,160,0,177,211,133,221,200
1170 DATA177,211,133,222,200,177,211,133
1180 DATA223,165,211,133,213,165,212,133
1190 DATA214,56,165,213,233,3,133,213
1200 DATA165,214,233,0,133,214,165,221
1210 DATA133,215,165,222,133,216,165,223
1220 DATA133,217,160,0,177,213,133,218
1230 DATA200,177,213,133,219,200,177,213
1240 DATA133,220,32,143,17,144,48,160
1250 DATA3,165,221,145,213,200,165,222
1260 DATA145,213,200,165,223,145,213,160
1270 DATA5,177,213,133,226,136,177,213
1280 DATA133,225,136,177,213,168,165,213
1290 DATA24,105,3,145,225,200,165,214
1300 DATA105,0,145,225,76,70,16,160
1310 DATA0,177,213,133,224,200,177,213
1320 DATA133,225,200,177,213,133,226,200
1330 DATA165,224,145,213,200,165,225,145
1340 DATA213,200,165,226,145,213,160,5
1350 DATA177,213,133,226,136,177,213,133
1360 DATA225,136,177,213,168,165,213,24
1370 DATA105,3,145,225,200,165,214,105
1380 DATA0,145,225,165,213,197,100,208
1390 DATA6,165,214,197,101,240,3,76
1400 DATA187,16,160,0,165,221,145,213
1410 DATA200,165,222,145,213,200,165,223
1420 DATA145,213,160,2,177,213,133,226
1430 DATA136,177,213,133,225,136,177,213
1440 DATA168,165,213,145,225,200,165,214
1450 DATA145,225,76,70,16,165,215,133
1460 DATA210,197,218,144,4,165,218,133
1470 DATA210,198,210,160,255,200,177,216
1480 DATA209,219,208,8,196,210,208,245
1490 DATA165,215,197,218,96
```

e ora esegui:

POKE 44,16: SAVE"....

Salvi la zona di memoria che va da $1001 alla fine del programma che hai appena scritto. In questo modo, quando carichi questo file e dai RUN ti trovi automaticamente con l'inizio del BASIC in $1201, con la routine caricata in memoria e il vettore di USR inizializzato; potrai quindi caricare un qualunque programma che potra' far uso di questa routine.
Un'ultima osservazione prima di passare alla descrizione del programma: se lo carichi quando e' allocata l'area di memoria per

la grafica non funziona, infatti il programma che carica i dati in memoria non si trova piu' in $1201 ma in $4201. Usando il comando GRAPHIC quando la routine e' gia' stata inizializzata non avrai nessun problema poiche' essa e' allocata nella zona di memoria inutilizzata in modo grafico (vedi Paragrafo 2.8 Figura 2.4).
L'algoritmo di ordinamento e'il seguente:

1- pone N=2
2- confronta l'elemento N con l'elemento N-1
3- se N>=N-1 pone N=N+1 e torna al punto 2
4- altrimenti:
   a- memorizza l'elemento di posizione N
   b- pone l'elemento N-1 nella posizione N
   c- confronta N con N-2
   d- se N-2>N pone N-2 nel posto N-1 e continua con N-3 ... fino a quando non trova un elemento N-I minore dell'elemento N (o fino ad arrivare al primo elemento da ordinare del vettore)
   e- pone l'elemento N nel posto N-I+1 (vedi figura 10.9)
   f- pone N=N+1
   g- torna al punto 2 fino ad aver esaurito tutti gli elementi del vettore.

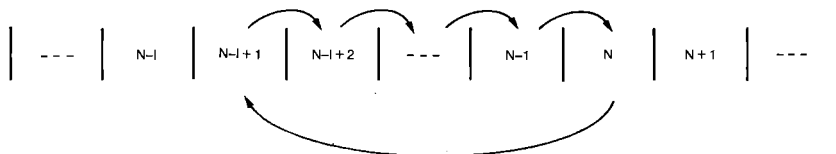



Figura 10.9 Algoritmo di ordinamento

Passiamo ora al commento della routine in linguaggio macchina:

```
MONITOR
   PC  SR AC XR YR SP
; 0000 00 00 00 00 F8
. 1022  78        SEI
. 1023  8D 3F FF  STA $FF3F
. 1026  20 2E 10  JSR $102E
. 1029  8D 3E FF  STA $FF3E
. 102C  58        CLI
. 102D  60        RTS
. 102E  A0 02     LDY #$02
. 1030  B1 5F     LDA ($5F),Y
. 1032  18        CLC
. 1033  65 5F     ADC $5F
. 1035  85 D0     STA $D0
. 1037  C8        INY
. 1038  B1 5F     LDA ($5F),Y
. 103A  65 60     ADC $60
. 103C  85 D1     STA $D1
. 103E  A5 64     LDA $64
. 1040  85 D3     STA $D3
. 1042  A5 65     LDA $65
```

```
1044  85 D4     STA $D4
1046  A5 D3     LDA $D3
1048  18        CLC
1049  69 03     ADC #$03
104B  85 D3     STA $D3
104D  A5 D4     LDA $D4
104F  69 00     ADC #$00
1051  85 D4     STA $D4
1053  C5 D1     CMP $D1
1055  90 0D     BCC $1064
1057  D0 06     BNE $105F
1059  A5 D3     LDA $D3
105B  C5 D0     CMP $D0
105D  90 05     BCC $1064
105F  A9 00     LDA #$00
1061  85 0D     STA $0D
1063  60        RTS
1064  A0 00     LDY #$00
1066  B1 D3     LDA ($D3),Y
1068  85 D7     STA $D7
106A  C8        INY
106B  B1 D3     LDA ($D3),Y
106D  85 D8     STA $D8
106F  C8        INY
1070  B1 D3     LDA ($D3),Y
1072  85 D9     STA $D9
1074  A5 D3     LDA $D3
1076  38        SEC
1077  E9 03     SBC #$03
1079  85 D3     STA $D3
107B  A5 D4     LDA $D4
107D  E9 00     SBC #$00
107F  85 D4     STA $D4
1081  A0 00     LDY #$00
1083  B1 D3     LDA ($D3),Y
1085  85 DA     STA $DA
1087  C8        INY
1088  B1 D3     LDA ($D3),Y
108A  85 DB     STA $DB
108C  C8        INY
108D  B1 D3     LDA ($D3),Y
108F  85 DC     STA $DC
1091  A5 D3     LDA $D3
1093  18        CLC
1094  69 03     ADC #$03
1096  85 D3     STA $D3
1098  A5 D4     LDA $D4
109A  69 00     ADC #$00
109C  85 D4     STA $D4
109E  20 8F 11  JSR $118F
10A1  B0 A3     BCS $1046
10A3  A0 00     LDY #$00
10A5  B1 D3     LDA ($D3),Y
10A7  85 DD     STA $DD
10A9  C8        INY
10AA  B1 D3     LDA ($D3),Y
10AC  85 DE     STA $DE
10AE  C8        INY
10AF  B1 D3     LDA ($D3),Y
10B1  85 DF     STA $DF
10B3  A5 D3     LDA $D3
10B5  85 D5     STA $D5
10B7  A5 D4     LDA $D4
10B9  85 D6     STA $D6
10BB  38        SEC
10BC  A5 D5     LDA $D5
10BE  E9 03     SBC #$03
10C0  85 D5     STA $D5
10C2  A5 D6     LDA $D6
10C4  E9 00     SBC #$00
10C6  85 D6     STA $D6
10C8  A5 DD     LDA $DD
10CA  85 D7     STA $D7
10CC  A5 DE     LDA $DE
10CE  85 D8     STA $D8
10D0  A5 DF     LDA $DF
10D2  85 D9     STA $D9
10D4  A0 00     LDY #$00
10D6  B1 D5     LDA ($D5),Y
10D8  85 DA     STA $DA
10DA  C8        INY
10DB  B1 D5     LDA ($D5),Y
10DD  85 DB     STA $DB
10DF  C8        INY
10E0  B1 D5     LDA ($D5),Y
10E2  85 DC     STA $DC
10E4  20 8F 11  JSR $118F
10E7  90 30     BCC $1119
10E9  A0 03     LDY #$03
10EB  A5 DD     LDA $DD
10ED  91 D5     STA ($D5),Y
10EF  C8        INY
10F0  A5 DE     LDA $DE
10F2  91 D5     STA ($D5),Y
10F4  C8        INY
10F5  A5 DF     LDA $DF
10F7  91 D5     STA ($D5),Y
10F9  A0 05     LDY #$05
10FB  B1 D5     LDA ($D5),Y
10FD  85 E2     STA $E2
10FF  88        DEY
1100  B1 D5     LDA ($D5),Y
1102  85 E1     STA $E1
1104  88        DEY
1105  B1 D5     LDA ($D5),Y
1107  A8        TAY
```

```
. 1108  A5 D5     LDA $D5
. 110A  18        CLC
. 110B  69 03     ADC #$03
. 110D  91 E1     STA ($E1),Y
. 110F  C8        INY
. 1110  A5 D6     LDA $D6
. 1112  69 00     ADC #$00
. 1114  91 E1     STA ($E1),Y
. 1116  4C 46 10  JMP $1046
. 1119  A0 00     LDY #$00
. 111B  B1 D5     LDA ($D5),Y
. 111D  85 E0     STA $E0
. 111F  C8        INY
. 1120  B1 D5     LDA ($D5),Y
. 1122  85 E1     STA $E1
. 1124  C8        INY
. 1125  B1 D5     LDA ($D5),Y
. 1127  85 E2     STA $E2
. 1129  C8        INY
. 112A  A5 E0     LDA $E0
. 112C  91 D5     STA ($D5),Y
. 112E  C8        INY
. 112F  A5 E1     LDA $E1
. 1131  91 D5     STA ($D5),Y
. 1133  C8        INY
. 1134  A5 E2     LDA $E2
. 1136  91 D5     STA ($D5),Y
. 1138  A0 05     LDY #$05
. 113A  B1 D5     LDA ($D5),Y
. 113C  85 E2     STA $E2
. 113E  88        DEY
. 113F  B1 D5     LDA ($D5),Y
. 1141  85 E1     STA $E1
. 1143  88        DEY
. 1144  B1 D5     LDA ($D5),Y
. 1146  A8        TAY
. 1147  A5 D5     LDA $D5
. 1149  18        CLC
. 114A  69 03     ADC #$03
. 114C  91 E1     STA ($E1),Y
. 114E  C8        INY
. 114F  A5 D6     LDA $D6
. 1151  69 00     ADC #$00
. 1153  91 E1     STA ($E1),Y
. 1155  A5 D5     LDA $D5
. 1157  C5 64     CMP $64
. 1159  D0 06     BNE $1161
. 115B  A5 D6     LDA $D6
. 115D  C5 65     CMP $65
. 115F  F0 03     BEQ $1164
. 1161  4C BB 10  JMP $10BB
. 1164  A0 00     LDY #$00
. 1166  A5 DD     LDA $DD
. 1168  91 D5     STA ($D5),Y
. 116A  C8        INY
. 116B  A5 DE     LDA $DE
. 116D  91 D5     STA ($D5),Y
. 116F  C8        INY
. 1170  A5 DF     LDA $DF
. 1172  91 D5     STA ($D5),Y
. 1174  A0 02     LDY #$02
. 1176  B1 D5     LDA ($D5),Y
. 1178  85 E2     STA $E2
. 117A  88        DEY
. 117B  B1 D5     LDA ($D5),Y
. 117D  85 E1     STA $E1
. 117F  88        DEY
. 1180  B1 D5     LDA ($D5),Y
. 1182  A8        TAY
. 1183  A5 D5     LDA $D5
. 1185  91 E1     STA ($E1),Y
. 1187  C8        INY
. 1188  A5 D6     LDA $D6
. 118A  91 E1     STA ($E1),Y
. 118C  4C 46 10  JMP $1046
. 118F  A5 D7     LDA $D7
. 1191  85 D2     STA $D2
. 1193  C5 DA     CMP $DA
. 1195  90 04     BCC $119B
. 1197  A5 DA     LDA $DA
. 1199  85 D2     STA $D2
. 119B  C6 D2     DEC $D2
. 119D  A0 FF     LDY #$FF
. 119F  C8        INY
. 11A0  B1 D8     LDA ($D8),Y
. 11A2  D1 DB     CMP ($DB),Y
. 11A4  D0 08     BNE $11AE
. 11A6  C4 D2     CPY $D2
. 11A8  D0 F5     BNE $119F
. 11AA  A5 D7     LDA $D7
. 11AC  C5 DA     CMP $DA
. 11AE  60        RTS
```

Le prime sei linee di programma selezionano la RAM da $8000 in poi in modo che il programma possa leggere le stringhe che vengono memorizzate in quella zona.
Il diagramma del programma e' il seguente:

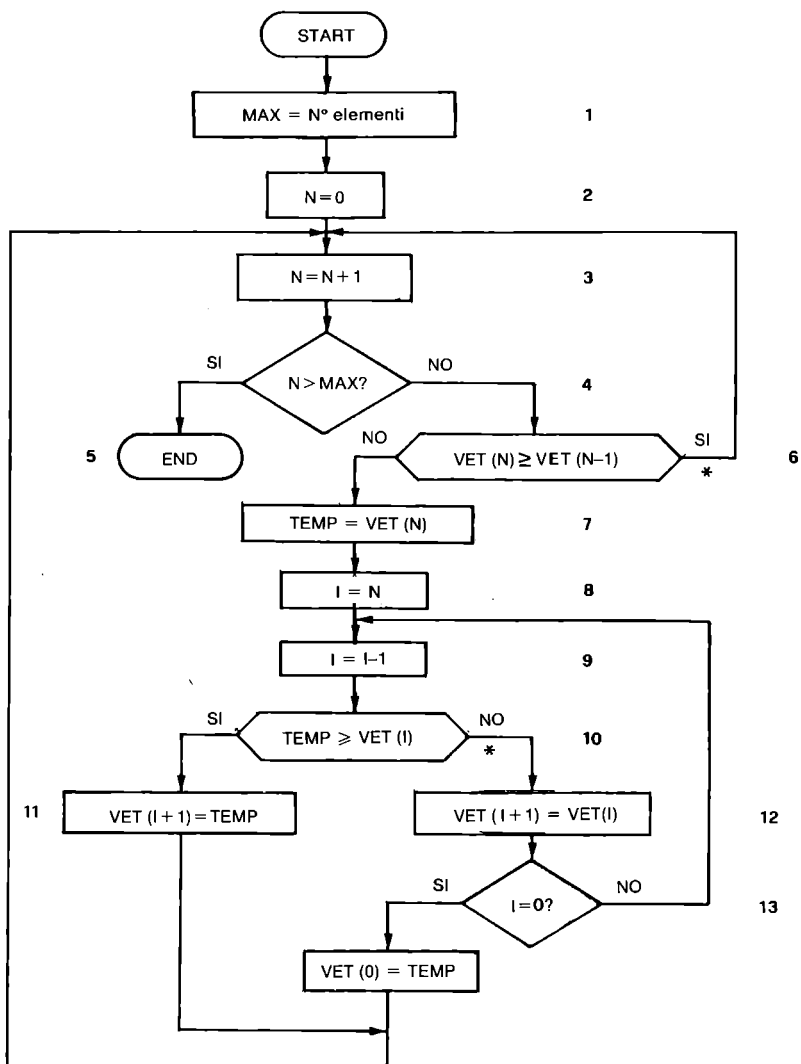



Figura 10.10 Schema a blocchi del programma ESEMPIO DI USR

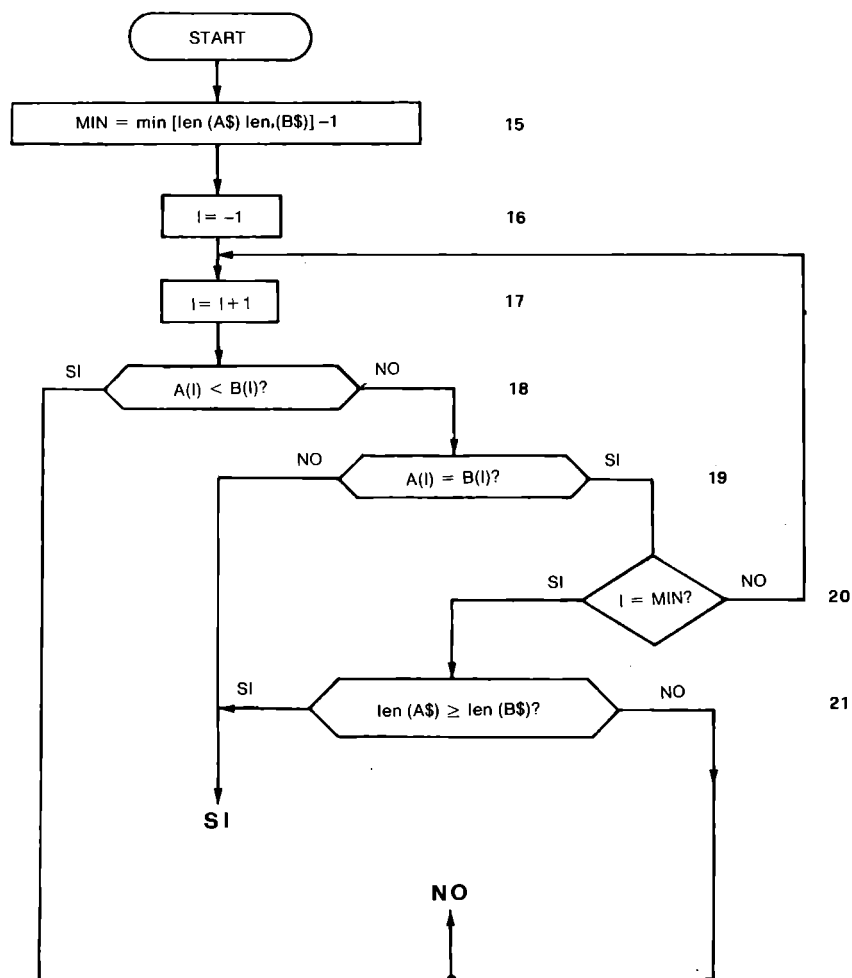



Figura 10.11 Schema a blocchi del sottoprogramma

Le linee da 1064 a 10A1 e da 10C8 a 10E7 pongono in sei byte di passaggio parametri, le lunghezze e i puntatori al primo carattere delle stringhe che la subroutine da 118F a 11AC deve confrontare; la routine tornera' i flag di zero e di carry alterati come li avrebbe alterati un'istruzione CMP.
Ed ecco infine la corrispondenza tra i blocchi del diagramma e le linee del programma:

BLOCCO 1 : linee 102E-103C
BLOCCO 2 : linee 103E-1044
BLOCCO 3 : linee 1046-1051
BLOCCO 4 : linee 1053-105D
BLOCCO 5 : linee 105F-1063 (prima di tornare avvisa
il BASIC che il risultato e' numerico)
BLOCCO 6 : linee 1064-10A1
BLOCCO 7 : linee 10A3-10B1
BLOCCO 8 : linee 10B3-10B9
BLOCCO 9 : linee 10BB-10C6
BLOCCO 10: linee 10C8-10E7
BLOCCO 11: linee 10E9-1116
BLOCCO 12: linee 1119-1153
BLOCCO 13: linee 1155-1161
BLOCCO 14: linee 1164-118C
BLOCCO 15: linee 118F-119B
BLOCCO 16: linee 119D
BLOCCO 17: linee 119F
BLOCCO 18-19: linee 11A0-11A4
BLOCCO 20: linee 11A6-11A8
BLOCCO 21: linee 11AA-11AC

IL BASIC 3.5

A.1 INTRODUZIONE

Il BASIC e' un linguaggio per programmare un calcolatore elettronico di tipo:

.CONVERSAZIONALE,
.INTERPRETATIVO.

Il termine CONVERSAZIONALE significa che durante la stesura del programma, la sua prova e la sua esecuzione, l'utente puo' intervenire per controllare risultati intermedi, apportare modifiche, correggere errori segnalati dal sistema, e tutto questo e' ottenibile con facilita' e immediatezza.
Il termine INTERPRETATIVO significa che nella memoria del calcolatore sono costantemente presenti, oltre al programma scritto in BASIC:
.un programma di sistema, che si chiama INTERPRETE BASIC,
.un insieme di programmi di sistema, che costituiscono il SISTEMA OPERATIVO,
e che essi interagiscono tra loro mettendo in grado l'utente di lavorare.
Il SISTEMA OPERATIVO e l'INTERPRETE BASIC costituiscono l'interfaccia tra l'utente e il calcolatore, che ha un suo linguaggio, il linguaggio macchina, e un suo modo di funzionare, che possono essere ignorati dall'utente.

Nel COMMODORE PLUS-4 il SISTEMA OPERATIVO e l'INTERPRETE BASIC risiedono stabilmente in una zona di memoria ROM di 32K byte.
Il BASIC del COMMODORE PLUS-4 e' una implementazione (versione) molto potente di questo linguaggio; essa e' siglata 3.5.

L'alfabeto del BASIC e' formato dai seguenti caratteri:

.le 26 lettere dell'alfabeto
A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z

.le 10 cifre decimali
0 1 2 3 4 5 6 7 8 9

.lo spazio

.19 caratteri speciali.
" # $ % π ( ) * + ,
¬ / : ; < = > ? ^

tutti presenti sulla tastiera.
Tutti i caratteri disponibili da tastiera, anche quelli non citati sopra, possono essere usati all'interno delle stringhe.

Gli elementi del linguaggio sono:

.COMANDI, .ISTRUZIONI,
.FUNZIONI, .OPERATORI,
.VARIABILI, .COSTANTI,

organizzati, secondo regole, in frasi o linee.

Le frasi del linguaggio possono essere scritte in due modi diversi:

.per ESECUZIONE IMMEDIATA,
.per ESECUZIONE DIFFERITA.

La distinzione viene fatta in base alla presenza di un numero all'inizio della linea; se non e' presente un numero vengono eseguiti subito, alla pressione del RETURN, i comandi e/o le istruzioni indicati. Se, invece, la linea inizia con un numero, essa viene memorizzata nell'area programma, inserendola al posto giusto in base al numero, per esecuzione differita.
Comunque la linea rappresenta un programma per il calcolatore, che puo' al limite consistere in un solo comando.
Qualora di una linea di programma facciano parte piu' comandi e/o istruzioni, essi devono essere separati dal carattere ":". La lunghezza di una linea di programma puo' essere di al massimo 88 caratteri.
I comandi, le istruzioni e le funzioni sono formate da PAROLE CHIAVE, che sono riservate, cioe' non possono essere usate per scopi diversi da quelli stabiliti, accompagnate, eventualmente, da parametri che ne precisano il significato.
Sono COMANDI quelle istruzioni che agiscono sulla preparazione e la verifica del programma, la sua memorizzazione su nastro o disco magnetico e il suo richiamo in memoria, la sua esecuzione, la situazione della memoria e la predisposizione dei tasti funzione. Sono ISTRUZIONI tutte le altre. Di norma i comandi ven-

gono usati in modo immediato, ma possono essere usati anche in modo differito, quando la cosa abbia senso logico. Non tutte le istruzioni possono essere usate in modo immediato.
Le FUNZIONI sono istruzioni particolari che forniscono un risultato numerico o stringa, o di stampa.

Una linea di programma e' formata dai seguenti elementi:
.NUMERO DI LINEA, da 0 a 63999, manca per esecuzione immediata,
.PAROLE CHIAVE (comandi, istruzioni, funzioni),
.COSTANTI,
.VARIABILI,
.OPERATORI,
.CARATTERI SEPARATORI.

I caratteri separatori ammessi sono:
.lo spazio (che puo' essere anche omesso, con lo svantaggio di rendere difficile la lettura, ma con il vantaggio di fare risparmiare memoria),
.i due punti (":"),
.la virgola (","),
.il punto e virgola (";").

A.2 COSTANTI E VARIABILI

Le COSTANTI possono essere:
.NUMERI INTERI, che vengono considerati tali solo se compresi tra -32767 e +32767. I numeri interi non compresi nell'intervallo citato vengono considerati reali.
.NUMERI REALI, che vengono stampati in forma decimale fino a 9 cifre (in valore assoluto minori di o uguali a 999999999, e non compresi nell'intervallo -0.1/+0.1, escluso lo zero), in forma esponenziale oltre.
.STRINGHE di caratteri alfanumerici, che sono in generale delimitate dal carattere virgolette. Esse non devono superare i 255 caratteri, ma, se fanno parte di una linea di programma, devono avere una lunghezza che non provochi il superamento degli 88 caratteri consentiti per la linea. Nel seguito viene indicato quando una stringa di caratteri puo' essere scritta senza virgolette delimitatrici.

Le costanti possono essere introdotte direttamente nelle linee di programma, oppure possono essere assegnate a delle variabili e richiamate mediante il nome delle variabili.
Nella memoria del calcolatore sono contenute due costanti speciali; una e' richiamata dai tasti
CBM-= ed e' π (π=3.14159265), l'altra si ottiene con la funzione EXP(1) ed e' il numero irrazionale "e", base dei logaritmi natu-

rali (e=2.71828183); queste costanti sono memorizzate con valore approssimato.

Le VARIABILI possono essere dei seguenti tipi:
.NUMERICHE INTERE,
.NUMERICHE REALI
.STRINGA
.NUMERICHE INTERE CON INDICE,
.NUMERICHE REALI CON INDICE,
.STRINGA CON INDICE.
Il tipo delle variabili viene evidenziato dal nome; le regole per la formazione dei nomi sono:
.formati da lettere o cifre,
.il primo carattere deve essere una lettera
.possono essere lunghi a piacere,
.vengono distinti in base ai primi 2 caratteri,
.devono terminare con il suffisso "%" per indicare numeri interi,
.devono terminare con il suffisso "$" per indicare stringhe,
.la mancanza di suffisso indica variabili numeriche reali,
.non possono far parte del nome le parole chiave del BASIC,
.la presenza di una coppia di parentesi dopo il nome con all'interno dei numeri o delle variabili numeriche, separati da virgole, indica che si tratta di variabili con indice del tipo specificato dal nome. Gli indici sono considerati solo per la parte intera, con troncamento degli eventuali decimali; essi partono da 0, cioe' il primo elemento ha indice 0.

Quando viene lanciato un programma con il comando RUN le variabili numeriche sono inizializzate al valore 0 e le variabili stringa alla stringa nulla. La stringa nulla e' una stringa che non contiene caratteri, di lunghezza 0; viene indicata da due virgolette vicine.
Le due stringhe A1$ e A2$ che seguono sono diverse:
A1$=" "   contiene uno spazio e ha lunghezza 1,
A2$=""   e' una stringa nulla di lunghezza 0.
Le variabili vengono riempite, cioe' contengono dati, per effetto di una delle seguenti operazioni:
.ASSEGNAZIONE, con o senza la parola chiave LET,
A$="BELLO", N=56.34, P%=6547, LET B$="piove".
.LETTURA, realizzata o con l'istruzione READ dall'interno del programma, o con le istruzioni GET, GETKEY e INPUT da tastiera, o con le istruzioni GET# e INPUT# da una periferica esterna.

Le variabili, non con indice, contengono un singolo dato; esse possono essere chiamate VARIABILI SINGOLE. Le VARIABILI CON INDICE sono riferite a un gruppo di dati, esse hanno tutte lo stesso nome e vengono distinte tra loro in base agli indici;

vengono chiamate anche ARRAY o MATRICI. Questa categoria di variabili risulta molto utile per trattare dati che appartengono a un insieme, che ha senso logico considerare globalmente. Il numero di indici da assegnare ad una variabile dipende dalle caratteristiche del gruppo di dati che la variabile deve rappresentare. Per una lista di dati basta un indice. Per dati raggruppati in una tabella bidimensionale, servono due indici, il primo rappresenta le righe, il secondo le colonne della tabella. Con 3 indici si possono trattare variabili visualizzabili in uno spazio a 3 dimensioni.
In questa implementazione del BASIC non esiste un limite teorico al numero di indici (dimensioni) assegnabili, in quanto questo puo' essere pari a 255, caso difficilmente verificabile. Inoltre, ogni dimensione puo' raggiungere il massimo degli interi positivi, cioe' 32767, cosa peraltro impossibile date le dimensioni della memoria del COMMODORE PLUS-4. Gli indici partono dal valore 0.
Esiste una istruzione per definire le variabili con indice, la DIM; essa deve essere usata per definire variabili per le quali almeno un indice superi il numero 10, cioe' gli 11 elementi. Per le variabili per le quali ogni indice raggiunge al massimo il valore 10, il BASIC provvede a una definizione implicita la prima volta che viene citata la variabile con indice.

Esempi di variabili:

BELLO numero reale, nome riconosciuto come BE
BESTIA numero reale, nome riconosciuto come BE
N% numero intero
B$ variabile stringa
C(23) numerica reale con indice, definita con DIM
B%(7) numerica intera con indice, puo' non essere definita con DIM (indice<11)
F(10,15,6) numerica reale con 3 indici, definita da DIM

Le VARIABILI BOOLEANE o LOGICHE sono variabili che nascono dall'esecuzione di particolari istruzioni nel corso del programma con il significato di VERO o FALSO; in realta', come vedremo, possono essere considerate di tipo aritmetico.

VARIABILI RISERVATE

Le variabili elencate nella Tabella B.4 sono ri servate, cioe' non possono essere usate nei programmi per scopi diversi da quelli loro assegnati dal sistema. Vediamo il loro significato.

DS e' la variabile di stato per le operazioni disco e serve per leggere dal canale comandi. Per ottenere un messaggio piu' leggibile, si puo' usare l'analoga variabile stringa DS$.

EL contiene il numero di linea dove si e' verificato un errore.
ER contiene il numero dell'ultimo errore che si e' verificato.
ERR$ e' il nome di una funzione che permette di vedere la descrizione dell'errore il cui numero e' in ER.

ST e' una variabile di stato che contiene dopo una operazione di INPUT o OUTPUT, non diretta alla tastiera o al video, un numero che con il suo valore da' notizia sull'esito dell'operazione stessa. Essa serve per riconoscere se l'operazione e' andata a buon fine, se si e' verificato un errore, o se si sono verificate particolari condizioni, come la fine di un file.

TI e TI$ si riferiscono all'orologio interno del COMMODORE PLUS-4.
TI viene azzerato al momento dell'accensione del calcolatore o del RESET; esso viene incrementato di 1 ogni 60-esimo di secondo, per cui con opportuni calcoli si puo' conoscere il tempo trascorso. TI puo' essere evidenziata con una istruzione di OUTPUT e il suo contenuto puo' essere trasferito in un'altra variabile, ma non puo' ricevere direttamente un valore.
TI$ e' una variabile stringa di 6 caratteri; essa contiene nei primi due caratteri le ore (da 0 a 23), nei successivi due i minuti (da 0 a 59) e negli ultimi due i secondi (da 0 a 59). Il suo contenuto dipende da TI, ma in essa si puo' scrivere per inizializzare l'orologio come si desidera. Quando si modifica TI$, viene automaticamente modificato TI. In conclusione TI$ puo' essere letta e scritta, TI puo' essere letta, ma scritta solo in modo indiretto tramite TI$. Quando TI$="235959" viene automaticamente azzerata TI e aggiornata TI$.

A.3 OPERATORI ARITMETICI, RELAZIONALI E LOGICI

Il BASIC consente di scrivere espressioni formate da COSTANTI e VARIABILI collegate tra loro da OPERATORI che possono essere di tipo ARITMETICO, RELAZIONALE, LOGICO.

Gli OPERATORI ARITMETICI sono:

+	somma (segno +)	-	sottrazione (segno -)
*	moltiplicazione	/	divisione
^	elevato a	(	aperta parentesi
)	chiusa parentesi		

Gli OPERATORI RELAZIONALI sono:

<	minore di	=	uguale a
>	maggiore di	<=	< di o = a
>=	> di o = a	<>	non uguale a

Gli OPERATORI LOGICI sono:

AND	e (l'uno e l'altro)
OR	o (l'uno o l'altro)
NOT	no (negazione)

Le espressioni vengono elaborate partendo da sinistra e andando verso destra, rispettando le regole di precedenza che seguono, riportate dalla piu' alta alla piu' bassa:

^	elevato a
-	segno meno
*/	moltipl. e div.
+-	somma e sottr.
>=<	relazione
NOT	negaz. logica
AND	AND logico
OR	OR logico

Se nell'espressione figurano delle parentesi, esse sono elaborate dando la precedenza a quelle piu' interne.

Nel calcolo delle espressioni numeriche ogni operazione tra due operandi da' come risultato un numero reale; alla fine del calcolo il risultato finale viene assegnato alla variabile che sta a sinistra dell'uguale rispettandone il tipo.

L'unica operazione aritmetica eseguibile tra stringhe e' la somma; essa consiste nell'unione delle due stringhe operando.

Le espressioni numeriche possono avere come operandi anche funzioni numeriche; analogamente le espressioni che danno come risultato una stringa possono avere come operandi funzioni stringa.

Gli operatori relazionali vengono usati per confrontare tra loro due espressioni; questo confronto ha significato numerico per espressioni numeriche, mentre ha un significato basato sull'ordinamento per espressioni stringa. Nel confronto tra stringhe vengono confrontati tra loro i codici ASCII dei caratteri componenti partendo da sinistra. I codici sono tali da mantenere il normale ordinamento alfabetico.
Dal confronto tra due operandi qualunque nasce una VARIABILE LOGICA, che puo' avere il significato di VERO o di FALSO, ma che ha un corrispondente significato aritmetico, -1 per VERO e 0 per FALSO. Questo fatto consente di introdurre in espressioni numeriche anche espressioni di tipo relazionale.
Esempi:

```
N%=R1$>R2$  N%=-1 se R1$>R2$
            N%= 0 se R1$=R2$ oppure R1$<R2$
```

A$="PIPPO" B$="PIPPO ", risulta A$<B$, infatti B$ contiene uno spazio dopo la parola PIPPO.

B=12345 B%=12345, risultano uguali anche se B e' una variabile reale e B% una variabile intera.

I confronti tra risultati numerici di calcoli complessi possono presentare piccole differenze dovute allo svolgimento dei calcoli in binario nella memoria del calcolatore.

Gli OPERATORI LOGICI, detti anche BOOLEANI, possono collegare tra loro due espressioni di qualunque tipo. L'applicazione di questi operatori produce un risultato logico di VERO o di FALSO in una variabile che, dal punto di vista aritmetico, vale 0 per FALSO e risulta diverso da 0 (non sempre -1) per VERO.

Gli operatori logici effettuano i confronti bit a bit secondo le regole che seguono:

Oper.logica	Risult.aritm.	Risult.relazion.
1 AND 1	1	VERO
1 AND 0	0	FALSO
0 AND 1	0	FALSO
0 AND 0	0	FALSO
1 OR 1	1	VERO
1 OR 0	1	VERO
0 OR 1	1	VERO
0 OR 0	0	FALSO
NOT 1	0	FALSO
NOT 0	1	VERO

Tutti gli operatori logici lavorano su espressioni numeriche, di cui considerano la parte intera; per non avere risultati errati la parte intera deve essere compresa tra -32767 e +32767.

A.4 COMANDI, ISTRUZIONI, FUNZIONI

In questo paragrafo elenchiamo in ordine alfabetico i comandi, le istruzioni e le funzioni del BASIC. Per ognuno di essi vengono indicate le parole chiave, gli eventuali parametri, il tipo (comando, istruzione, funzione), il modo (immediato o programma) nel quale puo' essere usato.
Nella spiegazione del linguaggio usiamo le seguenti convenzioni:

.Le PAROLE CHIAVE sono scritte in lettere maiuscole e non devono essere modificate (salvo le abbreviazioni riportate in Tabella B.2).
.I PARAMETRI (o ARGOMENTI) sono scritti in lettere minuscole.
.Le PARENTESI QUADRE delimitano argomenti opzionali.
.Le PARENTESI AD ANGOLO (simboli minore e maggiore) indicano che e' necessario scegliere uno degli argomenti presenti.
La BARRA INCLINATA separa argomenti tra i quali si puo' scegliere, senza introdurne altri.
I 3 PUNTINI indicano la possibilita' di ripetere piu' volte la sequenza.
Le VIRGOLETTE, se presenti devono essere mantenute.
Le PARENTESI ROTONDE se presenti devono essere mantenute.
Le abbreviazioni seguenti:

var	variabile,
esp	espressione,
numd	numero drive, numero disco nell'unita',
cond	condizione,
lfn	numero logico di un file,
nomef	nome file, abbreviato anche in fn,
unita'	numero logico periferica, abbreviato anche in dn.

Unita', numero logico periferica vale:
- 0 per la tastiera,
- 1 per il registratore,
- 2 per la linea RS 232
- 3 per il video.
- 4,5 per la stampante
- 6 per il plotter
- 8/11 per il disco

ABS Funzione numerica Immediato/Programma
ABS(esp) fornisce il valore assoluto di esp, che deve essere un'espressione numerica.

ASC Funzione numerica Immediato/Programma

ASC(x$) fornisce il codice ASCII del primo carattere della stringa contenuta nella variabile x$. L'argomento puo' essere una costante stringa tra virgolette. Il risultato e' un numero compreso tra 0 e 255.

ATN Funzione numerica Immediato/Programma
ATN(esp) fornisce l'angolo, espresso in radianti, che ha come tangente l'argomento, che deve essere un'espressione numerica. Per ottenere il risultato in gradi si deve procedere cosi': ATN(esp)*180/π.

AUTO Comando Immediato

AUTO [incremento] attiva la numerazione automatica delle linee di programma. Se manca incremento la numerazione automatica viene disattivata. L'incremento, che deve essere un numero intero, rappresenta la differenza tra due numeri di linea successivi. Il numero, da cui iniziare la numerazione automatica, dipende dal numero che viene attribuito alla linea di programma che si scrive subito dopo il comando AUTO. Quando si preme RETURN (cioe' viene accettata una linea di programma) compare automaticamente il numero della linea successiva.

AUTO 5 predispone l'incremento di 5 tra i numeri di linea.

AUTO annulla la predisposizione

Per fermare la numerazione basta rispondere con solo RETURN al numero di linea che compare, o con SHIFT-RETURN, se non si vuole cancellare la linea corrispondente.

BACKUP Comando Immediato/Programma

BACKUP Dnumd1 TO Dnumd2 [,ON Uunita']

duplica un dischetto su un secondo dischetto; puo' essere usato solo con un sistema collegato a una doppia unita' disco, e non 2 unita' singole. Il comando formatta il dischetto destinazione, che perde il suo precedente contenuto. Si ottiene una copia integrale del dischetto montato su numd1 (disco sorgente) nel dischetto montato su numd2 (disco destinazione).

BOX Istruzione Immediato/Programma

BOX [colore],x1,y1[,x2,y2][,angolo][,pit] disegna un rettangolo sul video, con le seguenti caratteristiche:

colore 0 o 1, default 1 (col. inchiostro) anche 2 e 3 in modo grafico multicolore

x1,y1 cordinate di un angolo

x2,y2 coordinate dell'angolo opposto

angolo rotazione in gradi (senso orario), default 0

pit pittura: 0 per OFF, 1 per ON (rettangolo colorato in colore); default 0

E' necessario porre delle virgole per segnalare gli argomenti intermedi omessi. Se e' presente solo una coppia di coordinate, l'angolo opposto e' la posizione del cursore grafico (pennino). Nel caso di una sola coppia di coordinate, dopo l'esecuzione dell'istruzione il cursore grafico non viene spostato dalla posizione precedente. Nel caso, invece, della presenza delle 2 coppie di coordinate il cursore grafico, dopo l'esecuzione dell'istruzione si trova nella posizione del secondo angolo specificato. L'istruzione da' risultati visibili solo se il calcolatore e' in uno dei modi grafici disponibili. Le coordinate possono essere influenzate dalla precedente esecuzione dell'istruzione SCALE.

CHAR Istruzione Immediato/Programma

CHAR[colore],x,y,"stringa"[,modo] consente di scrivere una stringa in una determinata posizione del video, le cui coordinate sono x,y, intese come posizione carattere (x da 0 a 39, y da 0 a 24). Puo' agire sia su un video predisposto per testo che per grafica. Solo che i caratteri di controllo eventualmente presenti nella stringa, gli stessi consentiti con l'istruzione PRINT, non agiscono nel video grafico. Nel video grafico+testo se Y>19 il COMMODORE PLUS-4 non mostra i caratteri richiesti, che pero' appaiono se si passa a un modo totalmente grafico.

colore 0 e 1, default 1 (col. inchiostro) anche 2 e 3 in modo grafico multicolore

x,y coordinate posizione iniziale

stringa testo da scrivere

modo 0 campo diretto, 1 campo inverso, validi solo per video grafico.

CHR$ Funzione stringa Immediato/Programma

CHR$(x) fornisce una stringa di un carattere, il cui codice ASCII e' x, che deve essere compreso tra 0 e 255. Se l'argomento non e' intero, ne viene considerata solo la parte intera. La stringa ha lunghezza 1 e puo' anche essere non stampabile.

CIRCLE Istruzione Immediato/Programma

CIRCLE[colore],[xc,yc],xr[,yr][,[ia][,[fa][,[angolo][,inc]]]]

disegna un cerchio, un ellissi, un arco o un poligono in dipendenza dai parametri presenti.

colore 0 e 1, default 1 (col. inchiostro) anche 2 e 3 in modo grafico multicolore

xc,yc coordinate del centro, default pos. del cursore grafico (pennino)

xr lunghezza orizzontale del raggio

yr lunghezza verticale del raggio, default xr, yr=xr per cerchi e poligoni regolari

ia posizione partenza arco in gradi, default 0

fa posizione finale arco in gradi, default 360

angolo rotazione in gradi in senso orario, default 0

inc ampiezza in gradi dell'angolo sotteso al lato del poligono regolare che approssima la circonferenza, default 2 gradi (180 lati)

L'istruzione agisce in modo visibile solo se e' stato selezionato un modo grafico. Se xr e yr sono diversi si ottengono ellissi. Per ottenere poligoni regolari si deve porre inc=(360/num.lati).

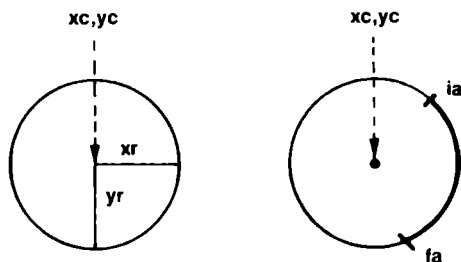



Figura A.1 Parametri CIRCLE

CLOSE Istruzione Immediato/Programma
CLOSE lfn chiude un file, di numero logico lfn, aperto con OPEN, su una periferica. L'operazione CLOSE deve sempre essere eseguita a conclusione delle operazioni che riguardano file, pena il buon funzionamento del sistema e la perdita di dati.

CLR Istruzione Immediato/Programma
CLR azzera tutte le variabili presenti in memoria, resettando i puntatori alle zone delle variabili ed esegue una RESTORE, ma non azzera il video. Inoltre chiude tutti i file, ma non in modo corretto, pulisce l'area STACK, e, per quest'ultima ragione non deve essere usato all'interno di cicli o sottoprogrammi. L'istruzione CLR viene eseguita automaticamente dopo: NEW, RUN, o operazioni di EDIT (modifica anche apparente del programma presente in memoria).

CMD Istruzione Immediato/Programma
CMD lfn [,lista] trasferisce l'uscita video su una periferica, sulla quale e' stato preventivamente aperto con OPEN il file logico lfn. Dopo CMD, PRINT e LIST agiscono sulla periferica selezionata, che puo' essere la stampante, il nastro o il disco. Dopo l'esecuzione delle operazioni di OUTPUT, per chiudere correttamente la comunicazione e' necessario eseguire i comandi:
PRINT #lfn:CLOSE lfn.

COLLECT Comando Immediato/Programma
COLLECT [Dnumd] [,ON Uunita']
sistema un dischetto dove sono presenti file non correttamente chiusi e aggiorna la DIRECTORY e la BAM. Esegue una VALIDATE del dischetto.

COLOR Istruzione Immediato/Programma
COLOR sorgente, colore [,luminosita'] consente di asse-

gnare un colore, mediante il suo numero a una delle 5 possibili sorgenti.
I numeri colore variano da 1 a 16, e sono quelli riportati sui relativi tasti numerici. Le possibili sorgenti sono numerate da 0 a 4, come segue:

Colore sfondo	Sorgente 0
Colore inchiostro	Sorgente 1
Modo multicolore 1	Sorgente 2
Modo multicolore 2	Sorgente 3
Colore bordo	Sorgente 4

La luminosita' puo' variare da 0 (bassa luminosita') a 7 (alta luminosita); il default e' 7. Non ha senso modificare la luminosita' del colore nero. Il richiamo del numero di una sorgente rende disponibile il colore che e' stato assegnato ad essa con COLOR. Alla partenza o dopo un RESET del calcolatore la situazione dei colori e' la seguente:

```
Sorg.0: col. 2, lum. 7   Sorg.1: col. 1, lum. 1
  "  2:  "  7   "  3       "  3:  "  12, lum. 5
  "  4:  "  15  "  6
```

CONT Comando Immediato

CONT fa ripartire un programma che si e' arrestato per effetto delle istruzioni STOP o END, o per l'uso del tasto RUN/STOP. Se il programma si e' fermato per l'istruzione END non compare alcun messaggio. Negli altri due casi compare il messaggio BREAK IN num-linea. CONT puo' essere usato solo se, dopo l'arresto del programma, non si e' entrati in EDITOR. Per entrare in EDITOR basta portare il cursore su una linea di programma e premere RETURN, anche senza fare alcuna modifica. E', invece, possibile usare comandi in immediato per visualizzare variabili, modificare il valore di alcune variabili, chiedere il LIST di parte del programma, e poi usare il comando CONT per proseguire.

COPY Comando Immediato/Programma

COPY [Dnumd1,] "nomef1" TO [Dnumd2,] "nomef2" [,ON Uunita']
copia il file di nome "nomef1" dal dischetto montato su numd1 nel file di nome "nomef2" sul dischetto montato su numd2. I dischetti possono essere diversi se e' collegata una doppia unita' disco, non 2 unita' singole. Se numd1=numd2 il comando produce una copia del file sullo stesso dischetto; in tale caso nomef1 deve essere diverso da nomef2.

COPY DO,"FILEA" TO D1,"FILEB"
copia il file "FILEA" dal dischetto montato sul drive 0 nel file "FILEB" sul dischetto montato sul drive 1.

COPY DO TO D1 copia tutti i file dal dischetto montato sul drive 0 al dischetto montato sul drive 1

COPY "PRIMO" TO "SECONDO"
copia sullo stesso dischetto il file "PRIMO" nel file "SECONDO".

COS   Funzione numerica   Immediato/Programma
COS (esp)   fornisce il coseno dell'argomento, che deve essere la misura di un angolo in radianti. Per trasformare i gradi in radianti basta moltiplicare per $\pi/180$.

DATA   Istruzione   Programma
DATA lista   consente di introdurre nel programma una serie di dati, che va a formare un file di dati interno al programma. La lista consiste in una serie di costanti di qualunque tipo, separate da virgola. Nel programma possono comparire diverse istruzioni DATA e possono essere posizionate dove si vuole. Quello che conta e' l'ordine con il quale le DATA si susseguono, infatti il file interno di dati viene formato secondo l'ordine delle frasi e l'ordine dei dati di ogni lista. Il programma al momento del RUN posiziona un PUNTATORE all'inizio del file di dati; esso viene spostato prima del dato successivo dopo la lettura di un dato con l'istruzione READ. Se si tenta di leggere piu' dati di quelli disponibili si ha segnalazione di errore. Esiste un'istruzione che consente di posizionare il puntatore all'inizio della lista dati di una particolare linea DATA; essa e' RESTORE.
Le costanti numeriche possono essere scritte in formato intero, con decimali e in formato esponenziale; le stringhe devono comparire tra virgolette solo se contengono i seguenti caratteri:
, : spazi SHIFT-lettera
caratteri grafici caratteri di controllo

DEC   Funzione numerica   Immediato/Programma
DEC(stringa)   fornisce il numero decimale corrispondente alla stringa, che deve essere una stringa, di al massimo 4 caratteri, rappresentante un numero esadecimale e puo' essere passata come una costante tra virgolette o come una variabile stringa.PRINT DEC("FF")   stampa 255
A$="5F":PRINT DEC(A$)   stampa 95

DEF FN   Istruzione   Programma
DEF FNnome(var)=esp   dove:
DEF FN puo' essere scritto anche DEFFN
nome   e' il nome che si vuole assegnare alla funzione e deve essere di 2 caratteri, con il primo alfabetico
var   e' il nome di una variabile reale che serve come argomento di comodo per la definizione matematica
esp   e' un'espressione di calcolo che puo' comprendere anche altre funzioni, ma solo di tipo reale

La funzione deve essere definita nel programma prima di richiamarla. Dopo la definizione essa si comporta come le funzioni del BASIC. Per richiamarla si deve citare il nome: FNnome(x), dove x e' l'argomento vero per il quale si vuole eseguire il calcolo, e puo' essere una variabile numerica reale o una costante.

DELETE   Comando   Immediato
DELETE [num1] [-num2] cancella le linee di programma BASIC dalla linea num1 alla linea num2.
DELETE 800   cancella la linea 800
DELETE 50-90   cancella dalla linea 50 alla linea 80
DELETE -100   cancella dall'inizio fino alla linea 100
DELETE 5000-   cancella dalla linea 5000 fino alla fine del programma

DIM   Istruzione   Immediato/Programma
DIM var(ind) [,var(ind)...]   dove var e' uno dei nomi consentiti per le variabili e tra parentesi sono indicati i valori massimi di ogni indice, separati da virgole. Gli indici possono essere costanti o variabili numeriche, delle quali viene considerata la parte intera.
Deve essere usata DIM per definire variabili con indice per le quali almeno un indice superi 10. Gli indici partono dal valore 0, quindi l'indice 10 corrisponde a 11 elementi. Fino a 11 elementi il sistema fa una definizione implicita della variabile al primo richiamo. In un programma non e' consentito dimensionare due volte la stessa variabile. Teoricamente il numero massimo di dimensioni puo' essere 255 e il limite per ogni indice e' 32767; bisogna ovviamente scegliere valori che siano compatibili con la memoria disponibile e con la lunghezza della linea di programma.

DIRECTORY   Comando   Immediato/Programma
DIRECTORY [Dnumd] [,Uunita'] [,"nomef"]   mostra la directory del dischetto montato sull'unita' collegata, senza distruggere il precedente contenuto della memoria. Puo' essere usato CTRL-S per fermare lo scrolling, che riprende alla pressione di un qualunque tasto. Per rallentare la visualizzazione si puo' premere il tasto CBM LOGO.
Per ottenere la copia su carta della DIRECTORY, se e' collegata una stampante, si deve procedere cosi:

```
LOAD"$0",8:OPEN4,4:CMD4:LIST
PRINT#4:CLOSE4
```

ma questa operazione distrugge il precedente contenuto della memoria.
DIRECTORY   lista la directory dell'unico dischetto collegato
DIRECTORY D1,U9,"ANAGRAFICO"
          lista la linea relativa al file di nome ANAGRAFICO della directory del dischetto montato sul drive 1 dell'unita' 9.

DLOAD Comando Immediato/Programma

DLOAD "nomef" [,Dnumd][,Uunita'] carica da disco il programma di nome "nomef". Puo' essere specificato il numero del drive, numd, necessario per unita' doppie; il numero dell'unita', che e' necessario nel caso di collegamenti multipli.
Invece del nome del programma, passato come costante stringa, puo' essere usata una variabile stringa inclusa tra parentesi:

```
N$="ANAGRAFICO":DLOAD(N$)
DLOAD"CONTABILITA'"
```

Se DLOAD e' usato da programma, dopo il caricamento il nuovo programma parte automaticamente, come per effetto di un GOTO alla prima linea. Devi fare attenzione perche' non viene riposizionato il puntatore all'inizio delle variabili (byte 45 e 46).
Per poter lavorare correttamente con programmi concatenati, ogni programma, prima di caricare il successivo, deve eseguire un CLR e scrivere in 45 e 46 i valori adatti.
Tali valori si rilevano caricando per prova il programma da concatenare e leggendo con la funzione PEEK il contenuto dei byte 45 e 46.
Un altro modo e' quello di scrivere come prima linea del programma richiamato:

```
0 POKE45,PEEK(157):POKE46,PEEK(158):CLR
```

che sistema il puntatore all'inizio delle variabili.

DO(LOOP) WHILE(UNTIL EXIT) Istruzione Immediato/Programma

DO [UNTIL cond/WHILE cond] Istruzioni[EXIT]
LOOP [UNTIL cond/WHILE cond] esegue le istruzioni comprese tra DO e LOOP, fino a quando sono soddisfatte le condizioni.
Se mancano UNTIL e WHILE, sia dopo DO che dopo LOOP, le istruzioni vengono eseguite indefinitamente. Se viene trovato EXIT, il ciclo viene interrotto e l'esecuzione prosegue dall'istruzione successiva a LOOP.
La condizione cond e' un'espressione relazionale o una variabile booleana; la parola chiave UNTIL significa "fino a quando la condizione diventa VERA, cioe' mentre e' FALSA", la parola chiave WHILE significa "fino a quando la condizione rimane VERA, termina appena diventa FALSA". Nella Figura A.2 sono riportati gli schemi logici.Il fatto di poter porre la condizione da analizzare dopo DO o dopo LOOP, da' una grande liberta' nell'uso dell'istruzione. Ponendo la condizione subito dopo DO, il ciclo puo' non essere mai eseguito, mentre ponendola dopo LOOP, si ha almeno una esecuzione.

```
10 DO UNTIL X=5 OR X=7    esegue le istruzioni
15 INPUT "X=";X           da 15 a 25 fino alla
20 ...                    lettura di 5 o 7
25 ...
30 LOOP
```

DO WHILE A$="":GET A$:LOOP   legge un carattere da tastiera fino a quando viene premuto un tasto, cioe' prosegue dopo la pressione di un tasto.

DRAW   Istruzione   Immediato/Programma
DRAW[colore][,x1,y1][,TO x2,y2][...]   consente di disegnare in modo visibile, se si e' in modo grafico, punti, linee e figure.
colore   0 e 1, default 1, (col. inchiostro) anche 2 e 3 in modo grafico multicolore
x1,y1   coordinate del primo punto, se mancano il default e' la posizione attuale del cursore grafico (PENNINO)
x2,y2   coordinate del secondo punto (finale) se manca TO x2,y2, viene disegnato solo un punto
Le coppie di punti iniziali e finali possono essere ripetute, ottenendo un disegno anche complicato. L'istruzione produce un effetto visibile solo se usata in modo grafico.

DSAVE   Comando   Immediato/Programma
DSAVE "nomef"[,Dnumd][,Uunita']   memorizza il programma presente in memoria sul disco, assegnandogli il nome "nomef". Per unita' doppie o collegamenti multipli devono essere specificati il numero del drive e quello dell'unita'. Puo' essere usata una variabile stringa al posto di "nomef", ma va inclusa tra parentesi.
DSAVE"PROVE"   memorizza il programma con il nome PROVE
N$="ORDIN":DSAVE(N$)   memorizza il programma con il nome ORDIN.

END   Istruzione   Programma
END   interrompe l'esecuzione del programma senza evidenziare messaggi. Si puo' proseguire scrivendo, se e' possibile, CONT e premendo RETURN.

EXP   Funzione numerica   Immediato/Programma
EXP(esp)   fornisce il valore della costante e (base dei logaritmi naturali, numero irrazionale di cui e' memorizzata un'approssimazione = 2.71828183) elevata al numero che risulta dal calcolo di esp. Se tale numero supera 88.0296919 si ha il messaggio di overflow.

FOR...TO...STEP   Istruzione   Immediato/Programma
FOR var=val-i TO val-f [STEP inc]   dove:
var   variabile numerica reale, detta variabile di controllo del ciclo
val-i   valore iniziale per var
val-f   valore finale per var
inc   incremento per var, puo' essere positivo o negativo; default 1

val-i, val-f e inc possono essere anche espressioni numeriche semplici e non avere valore intero. Se val-f e' minore di val-i, inc deve essere negativo.
L'istruzione serve per INIZIARE L'ESECUZIONE di un CICLO; essa non puo' essere usata da sola; al termine del ciclo deve comparire l'istruzione NEXT var (puo' essere scritta anche senza var, e allora si riferisce all'ultimo FOR eseguito).
Il ciclo e' composto da tutte le istruzioni comprese tra FOR e NEXT.
Vediamo cosa avviene:
.1) con l'esecuzione di FOR viene inizializzato il contatore var con il valore val-i e viene memorizzato il valore finale val-f e l'incremento;
.2) vengono eseguite tutte le istruzioni comprese tra FOR e NEXT;
.3) al NEXT viene sommato al contatore var l'incremento inc, e il valore viene confrontato con val-f;
.4) se il valore raggiunto non supera val-f (o non risulta minore per incremento negativo), il programma torna alla istruzione successiva al FOR (esegue un altro ciclo);
.5) se il valore raggiunto supera val-f (o e' inferiore) il programma prosegue con l'istruzione dopo il NEXT, cioe' esce dal ciclo.
All'uscita dal ciclo il valore di var e' superiore o inferiore a val-f. Inoltre il ciclo viene percorso almeno una volta, dato che la condizione viene analizzata al NEXT.
Se all'interno del ciclo sono presenti istruzioni condizionali (come IF) che fanno uscire dal ciclo, la variabile var deve essere forzata al valore finale e deve essere eseguito il NEXT, altrimenti il ciclo FOR resta pendente, non concluso, e questo puo' dar luogo a errori.

FOR K=K1TO K2
STEP K3 (K3>0)

DO WHILE COND
ISTR LOOP

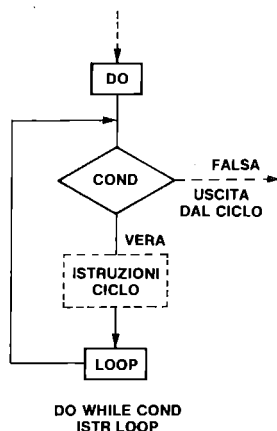
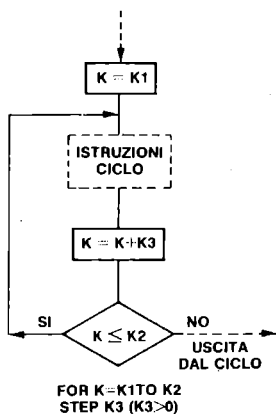
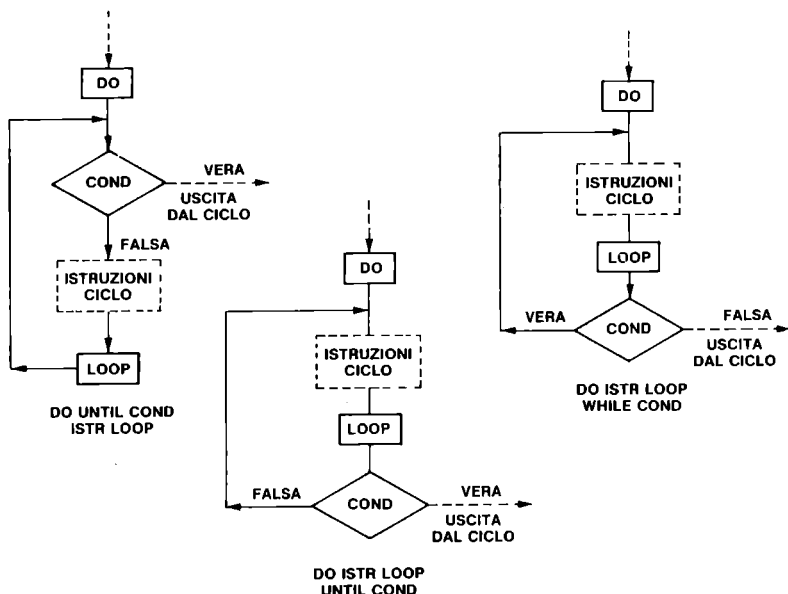



Figura A.2 Schemi logici FOR e DO...LOOP

E' possibile avere cicli FOR uno interno all'altro, si dice nidificati, ma non si possono avere intrecci, ogni ciclo deve essere completamente interno al precedente. Con il COMMODORE 16 si possono nidificare fino a 10 FOR.

FRE Funzione di servizio Immediato/Programma
FRE(x) fornisce il numero dei byte liberi in memoria, x puo' essere qualunque, si tratta di un argomento di comodo.

GET Istruzione Programma
GET lista variabili legge un carattere per volta dalla tastiera.
Se non e' stato premuto alcun tasto legge la stringa nulla. E' bene che lista contenga nomi di variabili stringa (per evitare errori), separate da virgole se piu' di una.
GET accetta un tasto senza che debba essere seguito dalla pressione di RETURN.
L'istruzione puo' essere usata per creare cicli di attesa fino alla pressione di un tasto qualunque o di un tasto particolare:

10 GETA$:IFA$=""THEN10   prosegue se si preme un tasto

10 GETA$:IFA$=<>"P"THEN10 prosegue se si preme P.

GETKEY   Istruzione   Programma
GETKEY lista variabili   e' simile a GET solo che prosegue solo se si preme un tasto.
10 GETKEYA$ prosegue se si preme un tasto
10 GETKEYA$:IFA$<>"P"THEN10 prosegue se si preme P
Non devi premere uno dei tasti funzione in risposta a GETKEY, infatti da' luogo a segnalazione di errore.

GET#   Istruzione   Programma
GET# lfn,lista variabili   dove:
lfn e' il numero logico di un file precedentemente aperto su una periferica
lista e' una lista di variabili separate da virgole
L'istruzione legge un carattere per variabile.
Con GET# si possono leggere tutti i caratteri, anche quelli di controllo.

GOSUB   Istruzione   Immediato/Programma
GOSUB num-linea   dove num-linea e' il nunero di una linea dove inizia un sottoprogramma.
Il controllo viene trasferito alla linea indicata, ma viene memorizzato il numero di linea e la posizione nella linea dell'istruzione GOSUB. Quando nel sottoprogramma viene incontrata l'istruzione RETURN, il controllo torna all'istruzione successiva a GOSUB.
All'interno di un sottoprogramma possono comparire altre istruzioni GOSUB; si dice che si ha il concatenamento dei sottoprogrammi. Si possono concatenare fino a 39 sottoprogrammi. Nella Figura A.3 e' riportato uno schema logico dell'istruzione.

GOTO   Istruzione   Immediato/Programma
GOTO num-linea oppure GO TO num-linea   fa proseguire il programma da num-linea, che deve esistere nel programma, pena segnalazione di errore.

GRAPHIC   Istruzione   Immediato/Programma
GRAPHIC modo [,flag] rende attivo per il video il modo grafico selezionato.
Flag puo' essere 1 per pulire il video, 0 per lasciarlo invariato.
I modi grafici sono:
0 testo
1 grafica ad alta risoluzione
2 grafica ad alta risoluzione e testo

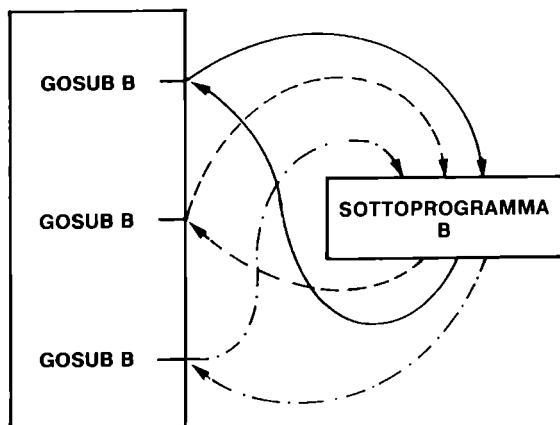



Figura A.3 Schema chiamata SOTTOPROGRAMMA

sulle 5 linee in basso
3 grafica multicolore
4 grafica multicolore e testo sulle 5
linee in basso

Quando si seleziona un modo da 1 a 4, il sistema assegna 12K byte alla grafica. (vedi Paragrafo 2.8). Anche se dopo essere passati da un modo grafico si esegue GRAPHIC 0, i 12K restano allocati, per renderli disponibili di nuovo si deve usare l'istruzione GRAPHIC CLR.

GSHAPE Istruzione Immediato/Programma
GSHAPE var¬stringa [,[x,y][,modo]] dove:
var¬stringa contiene l'immagine da mostrare
x,y sono le coordinate dell'angolo in alto a sinistra da dove iniziare, default la posizione del cursore
modo modo di visualizzazione:
0, senza modifiche (default)
1, campo inverso
2, con OR della zona
3, con AND della zona
4, con XOR della zona.

HEADER Comando Immediato/Programma
HEADER "nome¬disco",Dnumd [,Iid][,ON Uunita'] e' l'operazione di formattazione dei dischetti. Essa deve essere eseguita sui dischetti nuovi o su quelli che si vogliono rendere usabili ex-novo. Con HEADER il dischetto oltre a ricevere un nome e una identificazione, viene suddiviso in tracce e settori e vengono registrati i relativi indirizzi, e inoltre viene pre-

disposta una traccia per contenere l'indice dei contenuti (directory) e la tabella BAM di gestione dell'occupazione dei settori (blocchi). I parametri sono:
nome¬disco nome di al massimo 16 caratteri
numd numero del drive su cui e' montato il dischet¬to
id identificazione del dischetto, di 2 caratteri; e' necessaria per dischetti nuovi
unita' numero dell'unita'
Se per un disco gia' in uso si omette Iid, si ottiene di cancel¬lare il disco, ma vengono mantenuti gli indirizzi di traccia e settore e quindi l'operazione risulta piu' veloce.

HEX$ Funzione stringa Immediato/Programma
HEX$(n) fornisce una stringa di 4 caratteri che e' la rappresentazione esadecimale del numero n, che deve essere compreso tra 0 e 65535.

IF...THEN...ELSE Istruzione Programma
IF cond THEN Istruzioni1 [:ELSE Istruzioni2] dove cond e' una condizione.
Viene esaminata cond; se essa e' verificata, cioe' la condi¬zione risulta VERA viene eseguito il gruppo di istruzioni che segue THEN (Istruzioni1) e al loro termine il programma prosegue dalla linea seguente. Se, invece la condizione non e' verifi-cata, cioe' risulta FALSA si possono avere due casi: se ELSE e' presente vengono eseguite le istruzioni dopo ELSE (Istruzioni2) e poi il programma prosegue dalla linea seguente, se ELSE non e' presente il programma prosegue dalla linea seguente.
La parte ELSE, se presente deve trovarsi sulla stessa linea di IF...THEN. Se i gruppi di istruzioni dopo THEN e/o dopo ELSE contengono delle istruzioni tipo GOTO, il programma non va alla linea successiva a IF.

INPUT Istruzione Programma
INPUT ["messaggio";]lista variabili e' l'istruzione per leggere dati dalla tastiera. Le variabili di lista devono essere separate tra loro da virgole. Il sistema fa apparire un punto interrogativo di richiesta dati; si deve rispondere con tanti da¬ti quante sono le variabili, separandoli tra loro con virgole. Se il numero di dati e' inferiore al numero di variabili della lista il sistema richiede ancora dati con un doppio punto interro¬gativo. Se si risponde con dati di tipo che non concorda con il tipo delle variabili si ha un messaggio di errore e i dati ven¬gono chiesti di nuovo. L'immissione dei dati termina quando si preme RETURN. Se si risponde solo con RETURN, le variabili mantengono il loro precedente contenuto. Il messaggio tra virgo¬lette, se presente, viene evidenziato prima del punto interro-gativo di richiesta dati.

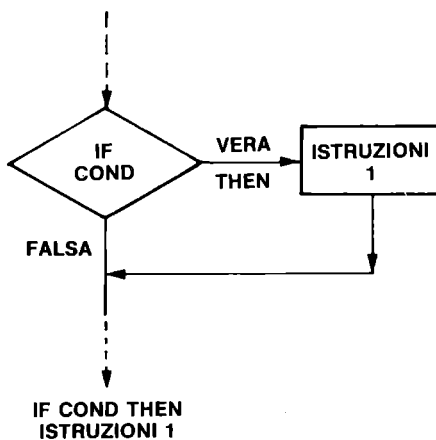
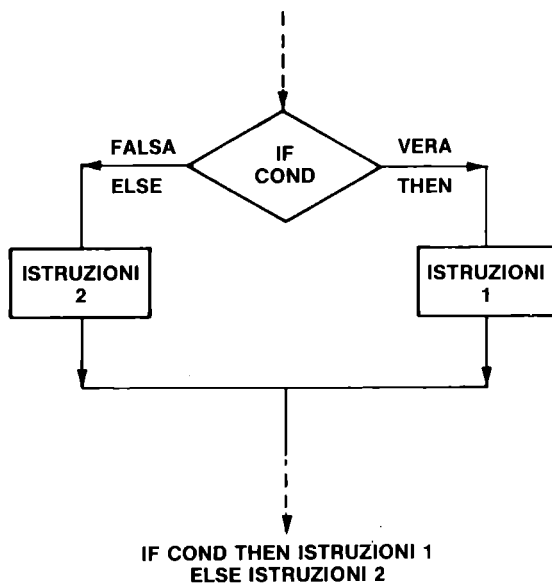



Figura A.4 Schema logico IF...THEN...ELSE

Se si risponde con piu' dati di quelli richiesti, quelli eccedenti vengono ignorati e il fatto viene segnalato.

INPUT#   Istruzione   Programma
INPUT# lfn, lista variabili   dove lfn e' il nume-

ro logico di un file precedentemente aperto su una periferica. L'istruzione e' simile alla precedente, solo che non puo' essere usato il messaggio, che d'altronde, non avrebbe senso.
L'istruzione preleva dati dal file considerando terminata una variabile quando incontra un separatore valido, che puo' essere la virgola (CHR$(44)) o il RETURN (CHR$(13)). Caratteri come "," e ":" all'interno di un dato stringa la fanno considerare terminata; per questo motivo le stringhe contenti questi caratteri devono essere scritte forzando all'inizio e alla fine le virgolette con CHR$(34), cosi':
PRINT#lfn,CHR$(34);"stringa";CHR$(34)
I dati separati tra loro da virgole, invece che da RETURN, devono essere letti da una sola istruzione INPUT#, che puo' agire solo per i dati contenuti tra due RETURN, e questi non devono superare 88 caratteri; in caso contrario si perdono dati.

INSTR Funzione numerica Immediato/Programma
INSTR(stringa1,stringa2[,pos-inizio]) ricerca in stringa1 la stringa2, partendo dall'inizio o da pos-inizio, se presente. Se non trova stringa2 da' risultato 0, se la trova da' la posizione di inizio di stringa2. Le due stringhe possono essere costanti o variabili.

INT Funzione numerica Immediato/Programma
INT(x) fornisce la parte intera di x troncando i decimali. Per arrotondare si deve operare cosi':
INT(x+0.5).

JOY Funzione numerica Immediato/Programma
JOY(n) fornisce la situazione di uno dei due joystick.
Se n=1, del joystick 1, se n=2 del joystick 2. Il numero fornito risulta >128 se e' stato premuto il bottone del fuoco e da' la direzione dello spostamento secondo lo schema seguente:

```
              alto
                1
              8   2
 sinistra  7    0    3 destra
              6   4
                5
              basso
```

alto+fuoo=129 destra+fuoco=131
numeri di posizione: 0, 1, 2, 3, 4, 5, 6, 7, 8.

KEY Comando Immediato/Programma
KEY [num-k,stringa] senza parametri fornisce l'elenco delle funzioni assegnate agli 8 tasti funzione (HELP corridsponde a F8). Con i parametri serve per assegnare una funzione a un tasto:

num-k deve variare da 1 a 8 e individua il tasto, stringa deve essere la serie di funzioni da assegnare al tasto sotto forma di stringa. Esempi:

```
KEY1,"OPEN4,4:CMD4:LIST"+CHR$(13)
KEY2,"PRINT#4:CLOSE4"+CHR$(13)
```

fanno si' che per listare un programma su stampante basta premere F1, e alla fine della stampa F2. Le assegnazioni iniziali sono:

```
KEY1,"SYS 1525: 3 PLUS 1"
KEY2,"DLOAD"+CHR$(34)
KEY3,"DIRECTORY"+CHR$(13)
KEY4,"SCNCLR"+CHR$(13)
KEY5,"DSAVE"+CHR$(34)
KEY6,"RUN"+CHR$(13)
KEY7,"LIST"+CHR$(13)
KEY8,"HELP"+CHR$(13).
```

LEFT$ Funzione stringa Immediato/Programma

LEFT$(stringa, num) dove:

stringa cost. o una var. stringa

num numero intero tra 0 e 255

fornisce i primi num caratteri della stringa.

Se num supera la lunghezza della stringa la fornisce tutta.

LEN Funzione numerica Immediato/Programma

LEN(stringa) dove stringa puo' essere una costante o una variabile. Fornisce il numero dei caratteri della stringa, 0 per la stringa nulla.

LET Istruzione Immediato/Programma

[LET] var=esp e' l'istruzione di assegnazione del BASIC. La parola chiave LET puo' mancare.

Il nome di var e il tipo dell'espressione devono concordare.

LIST Comando Immediato/Programma

LIST [numlinea1][-[numlinea2]] lista sul video una parte o tutto il programma presente in memoria. La pressione del tasto CBM LOGO rallenta lo scrolling; per arrestare la lista basta premere CTRL-S, poi la pressione di qualunque tasto fa continuare.

LIST senza parametri lista tutto il programma

LIST numlinea una sola linea

LIST numlinea1- da numlinea1 in avanti

LIST -numlinea2 fino a numlinea2

LIST numlinea1-numlinea2

il pezzo compreso tra i due numeri di linea.

LOAD Comando Immediato/Programma

LOAD ["nomef"[,unita'][,flag]] carica un programma da nastro o da disco. Dove:

nomef nome del programma e puo' anche essere una variabile stringa

unita' numero logico dell'unita': 1 per la cassetta e 8 per il disco

flag flag di rilocazione, vale 0 per rilocare (default), 1 per non rilocare (si veda comando SAVE)

LOAD carica il primo programma presente su nastro

LOAD "GIOCO",1 carica da cassetta il programma GIOCO

LOAD "PRIMO",8 carica da disco il programma PRIMO

Quando si usa da programma questo comando, dopo il caricamento il nuovo programma parte automaticamente, senza eseguire il CLR, quindi corrisponde a LOAD + GOTO prima linea.

Vedi il comando DLOAD per il problema della sistemazione dei byte 45 e 46.

LOCATE Istruzione Immediato/Programma

LOCATE x,y consente di posizionare il cursore grafico (pennino) in un punto di coordinate x,y; se ne puo' utilizzare l'effetto solo con video in modo grafico.

MID$ Funzione stringa Immediato/Programma

MID$(a$,n1,n2) puo' essere usata in due modi:

.1) per estrarre una parte di una stringa, se si trova a destra del carattere "=" in un'istruzione di assegnazione o in una lista di output,

.2) per modificare una parte di una stringa, se si trova a sinistra del carattere "=" in un'istruzione di assegnazione (pseudo variabile).

I parametri sono:

a$ stringa su cui operare, nel caso 1) puo' anche essere una espressione stringa o una costante

n1 cost. o var. numerica la cui parte intera da' la posizione da cui iniziare a operare

n2 cost. o var. numerica la cui parte intera da' il numero dei caratteri su cui operare

Il parametro n2 puo' essere omesso; allora nel caso 1) la funzione ritorna la parte destra della stringa a partire dal carattere di posizione n1. Nel caso 2) la presenza di n2 risulta inutile in quanto vengono sempre trattati i caratteri corrispondenti alla stringa sostitutiva.

```
100 A$="BELLA  GIORNATA":PRINT A$
105 MID$(A$,1)="BRUTTA":PRINT A$
```

sostituisce i caratteri da 1 a 6 della stringa A$ con "BRUTTA".

```
100 A$="OGGI PIOVE"
105 PRINT MID$(A$,6,5)
```
estrae "PIOVE" da A$ e lo stampa.

```
100 DATA BELLISSIMO,BRUTTISSIMO,INDIFFERENTI
105 FOR K=1 TO 3
110 READ A$:PRINT MID$(A$,1,5);" ";
115 NEXT K:PRINT
```
da' come risultato:
BELLI BRUTT INDIF

Nel caso di riassegnazione di parte di una stringa non deve venire modificata la lunghezza originale.

MONITOR   Istruzione   Immediato/Programma
MONITOR   consente di uscire dal BASIC e rende disponibile il programma MONITOR. Il MONITOR permette di lavorare in linguaggio macchina, cioe' e' possibile accedere alla memoria, disassemblare parti dell'INTERPRETE BASIC e del SISTEMA OPERATIVO, scrivere programmi in assembler, disassemblare programmi in linguaggio macchina, trasferire parti di memoria. Per tornare al BASIC basta scrivere X e premere RETURN.

NEW   Comando   Immediato/Programma
NEW   serve per cancellare il programma BASIC presente in memoria e tutte le sue variabili. Di norma si usa prima di cominciare a scrivere un nuovo programma. Se viene usato da programma, questo viene interrotto e cancellato.

NEXT   Istruzione   Immediato/Programma
NEXT [var,...,var]   deve essere usato dopo FOR...TO, si veda FOR.
Si puo' scrivere senza var, con solo una var, o con piu' var separate da virgole.

ON...GOTO (GOSUB) Istruzione   Programma
ON esp <GOTO/GOSUB> n1[,n2,...]   dove:
esp   e' un espressione numerica che deve dare un risultato non negativo, di cui viene considerata solo la parte intera
ni   sono i numeri di linea a cui trasferire il controllo
Se esp vale 1 il controllo e' trasferito a n1, se vale 2 a n2, e cosi' via. Se il valore di esp supera il numero dei numeri di linea presenti oppure e' 0, il programma prosegue dalla linea seguente ON. Se, invece, il valore di esp e' negativo si ha un messaggio di errore.

OPEN   Istruzione   Immediato/Programma

OPEN lfn [,unita'[,sa[,"nomef,tipo,modo"]]]   dove:

lfn   numero logico file, da 1 a 255

unita'   numero della periferica:
0 per tastiera   1 per cassetta
3 per video   4 per stampante
8 per disco

sa   indirizzo secondario che ha significato diverso a seconda delle periferiche; per la cassetta: 0 per leggere, 1 per scrivere, 2 per scrivere con segnalazione di fine-nastro, per il disco rappresenta il numero del canale, da 2 a 14, 15 per i comandi, per la stampante rappresenta il modo di stampa

nomef   nome del file di al massimo 16 caratteri

tipo   solo per i dischi: PRG, SEQ, REL, USR

modo   solo per i dischi: READ o WRITE.

OPEN 1,3   apre il video come Input o come Output

OPEN 2,0   apre la tastiera come Input

OPEN 1,1,0,"PIPPO"   apre la cassetta per leggere il file PIPPO

OPEN15,8,15   apre su disco il canale comandi

OPEN2,8,2,"ANAGRAFICO,SEQ,WRITE"   crea un file sequenziale su disco.

PAINT   Istruzione   Immediato/Programma

PAINT [colore] [,[x,y] [,modo]]   dove:

colore   0 o 1, default 1 (col. inchiostro) anche 2 e 3 in modo grafico multicolore

x,y   coordinate punto iniziale, default posizione cursore grafico (pennino)

modo   0, bordo del colore=1
1, bordo di colore diverso dallo sfondo

Serve per colorare parti di video delimitate da un bordo; il riempimento con il colore si arresta quando viene incontrato il bordo. Se il punto di partenza e' gia' del colore selezionato, non si ha pittura. Dopo l'esecuzione dell'istruzione il cursore grafico ritorna nel punto da cui e' partito per colorare. Produce effetti immediatamente visibili solo se usato in uno dei modi grafici. Le coordinate possono essere influenzate dalla precedente esecuzione di SCALE.

PEEK   Funzione numerica   Immediato/Programma

PEEK(n)   fornisce il contenuto del byte di indirizzo n, che puo' essere una costante o una variabile numerica, di cui interessa la parte intera, che deve essere compresa tra 0 e 65535.

POKE   Istruzione   Immediato/Programma

POKE n,x   scrive nel byte di indirizzo n il numero x. I limiti per n sono da 0 a 65535 e per x da 0 a 255. Sia n che x possono essere costanti o variabili numeriche di cui interessa la parte intera.

POS Funzione numerica Immediato/Programma
POS(x) dove x e' un argomento qualunque. Fornisce la posizione di colonna del cursore di testo, da 0 a 39.

PRINT Istruzione Immediato/Programma
PRINT [lista] dove lista e' una sequenza di elementi di stampa, che possono essere: costanti, variabili, funzioni, caratteri di controllo, separati tra loro da caratteri separatori validi. I caratteri separatori possono essere:
. Virgola, che ha l'effetto di mandare dopo la stampa alla prossima zona di stampa. Sul video le zone di stampa sono 4 per linea, ognuna di 10 caratteri.
. Punto e virgola, che non aggiunge spazi dopo la stampa.
. Spazio, che viene ignorato e non provoca aggiunta di spazi, ma puo' generare ambiguita' di interpretazione; per esempio: A B viene interpretato come una variabile, la AB e non come due variabili.
Se la lista di stampa termina senza punteggiatura (, o ;) si ottiene di andare a nuova linea dopo la stampa, cioe' l'assenza di punteggiatura corrisponde alla sequenza: CHR$(13); (cioe' RETURN, ma seguito dal punto e virgola).
Ogni elemento viene stampato con il suo formato, che per le stringhe e' il loro numero di caratteri, mentre per i numeri viene aggiunto uno spazio prima per il segno + o il segno meno, e uno spazio dopo. Una PRINT senza lista dati manda a nuova linea.
Questa istruzione e' di norma diretta al video; l'uso precedente di OPEN e di CMD puo' trasferire l'uscita a un'altra periferica; non viene in generale rispettata in questo caso la dimensione delle zone di stampa.
Le funzioni che influenzano il posizionamento per la stampa sono TAB, SPC.

PRINT# Istruzione Immediato/Programma
PRINT# lfn,lista dove lfn e' il numero logico di un file precedentemente aperto e lista e' una lista di elementi da scrivere. Per ogni periferica si possono avere differenze di comportamento riguardo ai codici di controllo, alle funzioni di posizionamento e ai caratteri separatori. E' importante rilevare che quando l'uscita e' su supporti magnetici, come nastri e dischi, tra gli elementi della lista devono comparire dei caratteri separatori, passati come stringa, in modo che vengano registrati e consentano di riconoscere, in fase di lettura, la fine delle variabili. I dati sono inviati in uscita carattere per carattere, l'incontro di un carattere separatore valido fa ritenere terminato il dato in fase di lettura. I caratteri separatori riconosciuti sono la virgola (CHR$(44)) e il RETURN (CHR$(13)) per tutti i tipi di dati; pero', se all'interno di una stringa si trova il carattere due punti o virgola, la stringa e'

considerata terminata, per evitare cio', stringhe di questa tipo devono essere registrate con il primo e l'ultimo carattere forzati a virgolette, con CHR$(34).

USING Istruzione Immediato/Programma
USING formato;lista viene usata in congiunzione con PRINT o PRINT# (PRINT USING...) e serve per modificare il formato dei da¬ti in uscita.
La lista e' la solita lista di dati da stampare: il formato e' una stringa di caratteri di controllo che svolgono specifiche azioni durante la stampa.
Il formato viene utilizzato per i dati della lista, eventual¬mente usandolo ripetutamente.
Segue l'elenco dei caratteri di controllo disponibili per i dati numerici:

predispone lo spazio per un carattere, se il dato eccede il numero di #, l'intero dato viene sostituito da tanti * quanti sono i #.

+ ¬ possono essere posti nella prima o nell'ultima posizione di un campo. Il + fa stampare il segno + o il segno ¬. Il - fa stampare solo il ¬ per i numeri negativi, per i positivi appare uno spazio. Se non si usa o il + o il -, il numero negativo viene preceduto dal ¬ occupando una delle posizioni previste per le cifre.

. definisce la posizione del punto decimale; se non e' stato previsto il punto decimale, il numero viene arrotondato a intero.

, consente di far apparire una virgola nella stessa posizione del numero; si possono predisporre piu' virgole e almeno un carattere # deve precedere la prima virgola. Nella stampa vengono evidenziate solo le virgole che hanno una cifra a sinistra, le altre sono ignorate. Se il numero e' negativo viene stampato il segno meno.

$ fa stampare il carattere $ dove indicato; se si vuole che il carattere $ si sposti e preceda la prima cifra significativa del numero, esso deve essere preceduto da un #. Se si usa anche + o ¬, il segno precede il carattere $ in stampa.

^^^^ fanno si che il numero venga stampato in formato esponen¬ziale. Se si usa anche # per definire il numero di caratteri, queste 4 frecce devono comparire dopo #. Per i numeri in formato esponenziale con caratteristica negativa si devono usare sempre ^^^^.

I caratteri di controllo per i campi stringa sono:

il numero di # determina il numero di caratteri della stringa da evidenziare, con eventuale troncamento a destra.

= serve per evidenziare una stringa al centro di un campo. Le dimensioni del campo sono date dal numero di # piu' 1, il

carattere = puo' comparire ovunque.

> consente di evidenziare la stringa allineata a destra nel campo. I caratteri # definiscono le dimensioni del campo, con eventuale troncamento a sinistra.

PUDEF Istruzione Immediato/Programma

PUDEF "stringa" consente di modificare 4 caratteri di controllo di USING; i seguenti: spazio, virgola, punto decimale e dollaro. Per rimpiazzare questi 4 caratteri con altri quattro si deve preparare una stringa che rechi nella prima posizione il carattere sostitutivo dello spazio, nella seconda quello della virgola, ecc.

PUDEF " .,\" consente di stampare i numeri nell'abituale modo italiano, con la virgola prima dei decimali, il punto a separare le migliaia e il carattere \ al posto del $.

RCLR Funzione numerica Immediato/Programma

RCLR(n) dove n e' il numero di una sorgente di colore, da 0 a 4. Fornisce il colore disponibile nella sorgente specificata.

RDOT Funzione numerica Immediato/Programma

RDOT(n) dove n vale 0 per coordinata x, 1 per coordinata y e 2 per sorgente colore. Fornisce informazioni sulla posizione del cursore grafico. Ha senso usarlo se e' stato prima reso attivo un modo grafico.

READ Istruzione Programma

READ lista consente di leggere i dati memorizzati nel file interno al programma con le istruzioni DATA. La lista e' formata da nomi di variabili separate da virgole. E' necessario che il tipo delle costanti concordi con il tipo delle variabili che vengono via via lette; altrimenti si ha segnalazione di errore. Si ha segnalazione di errore anche se si tenta di leggere piu' dati di quelli disponibili. Per ogni dato letto il puntatore interno nel file avanza di un dato.

REM Istruzione Programma

REM messaggio consente di introdurre commenti in un programma. Dopo REM il messaggio puo' contenere qualunque carattere. Questa istruzione deve essere o l'unica o l'ultima di una linea di programma.

RENAME Comando Immediato/Programma

RENAME [Dnumd,] "nomev" TO "nomen" [,Uunita'] consente di cambiare nome a un file su disco; "nomev" viene sostituito con "nomen" nella DIRECTORY del dischetto.

RENUMBER Comando Immediato
RENUMBER [numeron [,inc [,numerov]]] consente di rinumerare le linee di un programma partendo da numerov, che diventa numeron, e usando l'incremento inc. Se non si usano parametri la rinumerazione inizia con 10 e l'incremento e' 10. Vengono aggiornati tutti i richiami a numeri di linea presenti nel programma.
La struttura dell'istruzione consente rinumerazioni totali o parziali.

RESTORE Istruzione Programma
RESTORE [n] consente di riposizionare il puntatore interno ai dati memorizzati con le istruzioni DATA. Il riposizionamento avviene alla prima linea DATA, se non compare n, alla linea n, se presente questo parametro.

RESUME Istruzione Programma
RESUME [n/NEXT] consente di ritornare al programma dopo una routine di errore. Se si usa senza parametri il ritorno e' all'istruzione che ha causato l'errore. Con il parametro puo' essere alla linea n, oppure, con NEXT, all'istruzione successiva a quella che ha causato l'errore.

RETURN Istruzione Programma
RETURN deve essere presente alla fine logica (conclusione) di un sottoprogramma; quando viene eseguito, il controllo del programma ritorna alla istruzione seguente la GOSUB. Se si incontra RETURN, senza aver eseguito prima un GOSUB, si ha segnalazione di errore.

RGR Funzione numerica Immediato/Programma
RGR(x) dove x puo' essere qualunque, fornisce il numero del modo grafico corrente.

RIGHT$ Funzione stringa Immediato/Programma
RIGHT$(a$,x) fornisce gli x caratteri piu' a destra della stringa a$. Il numero x puo' variare tra 0 e 255; a$ puo' essere una qualunque espressione stringa.

RLUM Funzione numerica Immediato/Programma
RLUM(n) dove n e' il numero di una sorgente di colore, fornisce il grado di luminosita' della medesima, come numero.

RND Funzione numerica Immediato/Programma
RND(x) fornisce uun numero pseudo-casuale compreso tra 0 e 1, prelevandolo dalla sequenza generata dal calcolatore mediante un algoritmo pseudocasuale. La sequenza viene generata partendo da un numero iniziale chiamato "seme", che dipende dall'argomento:

x negativo, seme sempre allo stesso valore per lo stesso x; gli stessi numeri ogni volta che si usa il programma.

x=0, seme prelevato da TI; dipende dal tempo trascorso dall'accensione del calcolatore.

x positivo, la sequenza prosegue; i numeri dipendono dalla precedente inizializzazione.

RUN Comando Immediato/Programma

RUN [n] fa partire un programma, dall'inizio se manca n, dalla linea n se presente il parametro. Prima della partenza del programma viene eseguito un CLR.

SAVE Comando Immediato/Programma

SAVE["nomef"][,unita'][,flag]

memorizza il programma presente in memoria su nastro o su disco. I parametri sono:

nomef nome da assegnare al programma, puo' essere anche una variabile stringa

unita' numero logico periferica, 1 per la cassetta, 8 per il disco

flag 0 o assenza del flag, per memorizzare in modo rilocabile al momento del LOAD

1 per memorizzare mantenendo invariato il punto di caricamento

2 per il nastro per aggiungere EOT (fine nastro)

3 per il nastro per combinare gli effetti 1 e 2.

Per il nastro puo' mancare il nome, ma non e' consigliabile memorizzare senza un nome.

Se il flag viene omesso esso vale 0 e ha il significato di memorizzare il programma in modo rilocabile. La RILOCAZIONE di un programma ha questo significato:

. di norma il programma BASIC inizia al byte di indirizzo 4097 e questo indirizzo si trova nei byte 43 e 44 (puntatore all'inizio del programma),

. se prima di scrivere un programma sposti il puntatore all'inizio del programma modificando il contenuto dei byte 43 e 44, il programma inizia a un indirizzo diverso da 4097,

. se usi il flag=0 nell'istruzione SAVE il programma viene memorizzato in modo rilocabile,

. se usi il flag=1 o il flag=3 nell'istruzione SAVE il programma viene memorizzato mantenendo l'informazione sul suo indirizzo di inizio, qualunque esso sia, e non e' rilocabile,

. al momento del LOAD, istruzione nella quale e' possibile usare per il flag il valore 1, o il valore 0 (assenza di flag), si ha il seguente comportamento:

.. per flag=0 il programma viene caricato a partire dall'indirizzo di inizio programma che si trova nei byte 43 e 44, rilocandolo o meno a seconda del modo come era stato memorizzato;

.. per flag=1 il programma viene caricato a partire dall'indirizzo dove si trovava al momento del SAVE, e questo puo' essere in disaccordo con il valore contenuto nei byte 43 e 44.
Se per memorizzare e' stato usate il flag, l'istruzione VERIFY deve essere scritta in modo tale che non ci sia contrasto con la situazione della memoria e quello che sta sul nastro.

```
SAVE "PROVE"        memorizza su nastro con nome PROVE
SAVE                memorizza su nastro senza nome
SAVE "PROVA",8      memorizza su disco con nome PROVA
A$="MOVI":SAVE A$,8         memorizza su disco con nome MOVI
SAVE "ANALISI",1,2          memorizza su nastro con EOT finale.
```

SCALE   Istruzione   Immediato/Programma
SCALE <1/0>   ha un effetto immediatamente visibile solo se usata in uno dei modi grafici; permette di introdurre un fattore di scala per le coordinate.
Di norma, o per effetto di SCALE 0, i limiti per le coordinate grafiche sono:
.in modo alta risoluzione 0<=x<=319 e 0<=y<=199
.in modo multicolore   0<=x<=159 e 0<=y<=199
Dopo l'esecuzione di SCALE 1 le coordinate x e y possono variare da 0 a 1023. Questo significa che vengono accettati per x e y valori nel range 0-1023 e pensa il sistema ad aggiustare i valori per far entrare i punti nel quadro video.

SCNCLR   Istruzione   Immediato/Programma
SCNCLR   pulisce lo schermo anche in modo grafico.

SCRATCH   Comando   Immediato/Programma
SCRATCH "nomef"[,Dnumd][,Uunita']   cancella il file di nome "nomef" dalla directory del dischetto montato sul drive numd della periferica di numero logico unita'. Se mancano gli ultimi 2 parametri deve essere collegata una unita' a disco singolo. Per evitare cancellazioni involontarie il sistema chiede conferma con il messaggio "ARE YOU SURE?"; rispondendo Y avviene la cancellazione, rispondendo N no.

SGN   Funzione numerica   Immediato/Programma
SGN(esp)   dove esp deve essere un'espressione numerica. Fornisce il segno dell'espressione: 0 se esp vale 0, 1 se e' positiva, -1 se e' negativa.

SIN   Funzione numerica   Immediato/Programma
SIN(x)   fornisce il seno di x, che e' espresso in radianti.

SOUND   Istruzione   Immediato/Programma
SOUND voce, frequenza, durata   produce un suono usan-

do una delle due voci disponibili, con una frequenza che puo' variare da 0 a 1023 e con una durata, in 60-esimi di secondo, che puo' variare da 0 a 65535.
Voce puo' valere: 1 per la voce 1
2 per la voce 2
3 per il rumore bianco della voce 2
Nell'esecuzione di due comandi SOUND successivi con voce uguale, il secondo inizia quando il primo suono e' terminato; ponendo la durata 0 si ottiene di far cessare immediatamente il suono selezionato.

SPC Funzione di stampa Immediato/Programma
SPC(esp) dove esp deve dare un risultato numerico compreso tra 0 e 255, di cui viene considerata solo la parte intera.
La funzione provoca il salto di n posizioni, se n e' il valore di esp, anche con passaggio alle linee successive, cioe' corrisponde all'invio di n caratteri "cursore a destra". L'uso di SPC consente di ottenere incolonnamenti, tenendo conto anche della lunghezza dei dati.

SQR Funzione numerica Immediato/Programma
SQR(esp) dove esp deve essere un'espressione numerica maggiore o uguale a zero. Fornisce la radice quadrata di esp.

SSHAPE Istruzione Immediato/Programma
SSHAPE var,x1,y1 [,x2,y2] dove:
var e' il nome di una variabile stringa
x1,y1 sono le coordinate di un angolo della zona video da memorizzare
x2,y2 sono le coordinate dell'angolo opposto
L'istruzione, che ha un effetto immediatamente visibile solo se usata in uno dei modi grafici, consente di memorizzare zone rettangolari del video in una stringa. E' necessario calcolare le dimensioni che deve assumere la stringa, che non puo' superare i 255 caratteri, e, in caso, delimitare la zona o frazionarla per memorizzarla in piu' strighe. Per calcolare le dimensioni si possono usare le formule che seguono:

per l'alta risoluzione
INT((ABS(x1-x2)+1)/4+.99)*(ABS(y1-y2)+1)+4
per il multicolore
INT((ABS(x1-x2)+1)/8+.99)*(ABS(y1-y2)+1)+4

La memorizzazione avviene riga per riga; negli ultimi 4 byte della stringa si trovano le lunghezze delle colonne e delle righe del rettangolo, nel formato byte basso-byte alto. Tali lunghezze, se e' attivo SCALE, devono essere divise rispettivamente per 5.12 (colonne) e 3.2 (righe).

STEP Istruzione Immediato/Programma

STEP esp dove esp deve essere un'espressione numerica e rappresenta l'incremento da usare nel controllo di un ciclo con l'istruzione FOR. Si veda FOR.

STOP Istruzione Immediato/Programma

STOP ferma il programma e compare il messaggio indicante il numero di linea della STOP. Ha lo stesso effetto della pressione del tasto RUN/STOP. Per continuare premere CONT, se e' possibile continuare.

STR$ Funzione stringa Immediato/Programma

STR$(esp) dove esp deve essere un'espressione numerica. Fornisce la stringa corrispondente al numero risultato di esp. La stringa generata contiene all'inizio o uno spazio o il segno meno.

SYS Istruzione Immediato/Programma

SYS in dove ind e' un indirizzo di memoria (compreso tra 0 e 65535). All'indirizzo indicato deve esserci un sottoprogramma in linguaggio macchina che va in esecuzione; esso deve terminare con il comando RTS che fa proseguire il programma BASIC dall'istruzione successiva alla SYS. A differenza di USR, con SYS non si passano parametri; per questa ragione i risultati del programma in linguaggio macchina devono essere memorizzati in posizioni di memoria definite in modo opportuno, e poi da li' prelevati con la funzione PEEK.

TAB Funzione di stampa Immediato/Programma

TAB(esp) dove esp deve fornire un valore numerico compreso tra 0 e 255. Per il video esp rappresenta la posizione di stampa dove portare il cursore contando dall'inizio della linea; se la posizione e' gia' stata superata dal cursore, essa non agisce; se la posizione e' fuori dalla linea, prosegue sulle linee seguenti. Quando la stampa e' diretta alla stampante la funzione TAB agisce come la funzione SPC, cioe' il conto della posizione e' relativo alla posizione attuale.

TAN Funzione numerica Immediato/Programma

TAN(x) fornisce la tangente dell'angolo x, che deve essere espresso in radianti.

TO Istruzione Immediato/Programma

TO fa parte delle istruzioni: BACKUP, COPY, DRAW,FOR, GO TO, e a queste si rimanda.

TRAP Istruzione Immediato/Programma

TRAP [n] consente di rendere attiva una routine di errore, scritta dall'utente a partire dalla linea n. In tale routine possono essere analizzate le variabili EL, ER, ERR$ per conoscere quale errore si e' verificato ed agire in conseguenza. Per uscire dalla routine di errore si deve usare l'istruzione RESUME. Quando e' stata eseguita l'istruzione TRAP n, va in esecuzione la routine di errore per tutti gli errori salvo quello di codice 17 "UNDEF'D STATEMENT ERROR". Va in esecuzione la routine anche se si preme il tasto RUN/STOP.
L'uso di TRAP senza parametro disabilita la funzione.

TROFF Istruzione Immediato/Programma

TROFF disabilita la funzione TRON e non vengono piu' listati i numeri delle linee di programma eseguite.

TRON Istruzione Immediato/Programma

TRON abilita la stampa dei numeri di linea eseguiti dal programma. E' molto utile per trovare errori nei programmi. I numeri di linea compaiono tra parentesi quadre.

UNTIL Istruzione Immediato/Programma

UNTIL fa parte dell'istruzione DO...LOOP, a cui si rimanda.

USR Funzione numerica Immediato/Programma

USR(x) consente di andare ad eseguire un programma in linguaggio macchina, il cui indirizzo di inizio deve essere stato precedentemente memorizzato nei byte 1281 e 1282 (byte basso-byte alto). Il parametro x viene passato al programma in linguaggio macchina nell'accumulatore 1; esso e' formato da una serie di 5 byte da 97 a 102, cosi' utilizzati: 97 per l'esponente, 98-101 per la mantissa e 102 per il segno. Se il parametro e', invece, una stringa: 97 per il numero dei caratteri, 98-99 per il puntatore al primo carattere del corpo della stringa. In tale accumulatore si trova un eventuale risultato, che pero' viene anche reso disponibile al programma BASIC come valore della funzione. Il programma in linguaggio macchina deve terminare con l'istruzione RTS.

VAL Funzione numerica Immediato/Programma

VAL(a$) dove a$ puo' essere una costante o una variabile stringa e deve essere di contenuto numerico. La funzione trasforma la stringa in un numero. La conversione si ferma al primo carattere non numerico incontrato. Se il primo carattere della stringa non e' numerico (+, - o cifra) il risultato e' 0.

VERIFY Comando Immediato/Programma

VERIFY "nomef"[,unita'][,flag] confronta il programma

presente in memoria con quello selezionato o su disco o su nastro. E' buona norma eseguire la verifica dei programmi dopo averli memorizzati. Nel caso della cassetta questa istruzione puo' essere usata per far avanzare il nastro fino a dopo l'ultimo programma memorizzato; basta usare VERIFY con quel nome e naturalmente il risultato non sara' O.K., ma il nastro sara' arrivato al punto giusto. Il flag deve essere presente, quando necessario, se il programma da verificare e' stato memorizzato con il flag; a questo proposito vale quanto detto per il flag nella descrizione del comando SAVE.

VOL Istruzione Immediato/Programma

VOL n predispone il volume per il comando SOUND. Il numero n puo' variare tra 0 e 8. Il valore 0 annulla il volume. Il volume e' il medesimo per le due voci.

WAIT Istruzione Immediato/Programma

WAIT n,x1 [,x2] dove:

n e' un indirizzo di memoria, da 0 a 65535
x1 e' un numero tra 0 e 255
x2 e' un numero tra 0 e 255

L'operazione agisce cosi':

. viene analizzato il contenuto del byte n,
. viene eseguita una operazione di OR-esclusivo (XOR) tra il contenuto del byte di indirizzo n e il numero x2; se x2 manca viene assunto uguale a 0,
. viene eseguita una operazione AND tra il risultato precedente e x1,
. se il risultato finale e' 0, cioe' FALSO, viene eseguita nuovamente la WAIT,
. se il risultato e' diverso da zero, cioe' VERO, il programma prosegue.

Con l'uso non appropriato di questa istruzione si possono produrre dei cicli infiniti.
Le regole delle operazioni logiche AND e XOR sono le seguenti:

```
1 AND 1 = 1    1 AND 0 = 0
0 AND 1 = 0    0 AND 0 = 0

1 XOR 1 = 0    1 XOR 0 = 1
0 XOR 1 = 1    0 XOR 0 = 0
```

WHILE Istruzione Immediato/Programma

WHILE fa parte dell'istruzione DO...LOOP, a cui si rimanda.

ISTRUZIONI ASSEMBLER

Nelle tabelle delle pagine seguenti sono riportate le istruzioni ASSEMBLER.
Ogni riga riguarda un'istruzione, riportata in ordine alfabetico per codice mnemonico. Per ogni istruzione viene data in colonna 2 la descrizione sintetica dell'operazione. Seguono sulle colonne successive, per ogni tipo di indirizzamento, il codice esadecimale dell'istruzione, il numero dei cicli di clock impiegati per eseguirla e il numero dei byte occupati. Nelle ultime 6 colonne sono indicati i FLAG influenzati dall'esecuzione dell'istruzione.

ISTRUZIONI		immediato			assoluto			pag zero			accum			implicito		
mnemo	operazione	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#
ADC	A←A+M+C * ***	69	2	2	6D	4	3	65	3	2						
AND	A←A∧M *	29	2	2	2D	4	3	25	3	2						
ASL	C←[7 0]←0				0E	6	3	06	5	2	0A	2	1			
BCC	salta se C=0**															
BCS	salta se C=1**															
BEQ	salta se Z=1**															
BIT	A∧M				2C	4	3	24	3	2						
BMI	salta se N=1**															
BNE	salta se Z=0**															
BPL	salta se N=0**															
BRK	vedi cap 5.4													00	7	1
BVC	salta se V=0**															
BVS	salta se V=1**															
CLC	C←0													18	2	1
CLD	D←0													D8	2	1
CLI	I←0													58	2	1
CLV	V←0													B8	2	1
CMP	A-M *	C9	2	2	CD	4	3	C5	3	2						
CPX	X-M	E0	2	2	EC	4	3	E4	3	2						
CPY	Y-M	C0	2	2	CC	4	3	C4	3	2						
DEC	M←M-1				CE	6	3	C6	5	2						
DEX	X←X-1													CA	2	1
DEY	Y←Y-1													88	2	1
EOR	A←A⊻M *	49	2	2	4D	4	3	45	3	2						
INC	M←M+1				EE	6	3	E6	5	2						
INX	X←X+1													E8	2	1
INY	Y←Y+1													C8	2	1
JMP	salta				4C	3	3									
JSR	salta a sotprg				20	6	3									
LDA	A←M *	A9	2	2	AD	4	3	A5	3	2						

	(ind,X)			(ind),Y			pag 0,X			assol,X			assol,Y		
mnemo	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#
ADC	61	6	2	71	5	2	75	4	2	7D	4	3	79	4	3
AND	21	6	2	31	5	2	35	4	2	3D	4	3	39	4	3
ASL							16	6	2	1E	7	3			
BCC															
BCS															
BEQ															
BIT															
BMI															
BNE															
BPL															
BRK															
BVC															
BVS															
CLC															
CLD															
CLI															
CLV															
CMP	C1	6	2	D1	5	2	D5	4	2	DD	4	3	D9	4	3
CPX															
CPY															
DEC							D6	6	2	DE	7	3			
DEX															
DEY															
EOR	41	6	2	51	5	2	55	4	2	5D	4	3	59	4	3
INC							F6	6	2	FE	7	3			
INX															
INY															
JMP															
JSR															
LDA	A1	6	2	B1	5	2	B5	4	2	BD	4	3	B9	4	3

	relativo			indiretto			pag 0,Y			FLAG					
mnemo	CO	N	#	CO	N	#	CO	N	#	N	Z	C	I	D	V
ADC										X	X	X			X
AND										X	X				X
ASL										X	X	X			
BCC	90	2	2												
BCS	B0	2	2												
BEQ	F0	2	2												
BIT										M7	X				M6
BMI	30	2	2												
BNE	D0	2	2												
BPL	10	2	2												
BRK													1		
BVC	50	2	2												
BVS	70	2	2												
CLC												0			
CLD														0	
CLI													0		
CLV															0
CMP										X	X	X			
CPX										X	X	X			
CPY										X	X	X			
DEC										X	X				
DEX										X	X				
DEY										X	X				
EOR										X	X				
INC										X	X				
INX										X	X				
INY										X	X				
JMP				6C	5	3									
JSR															
LDA										X	X				

ISTRUZIONI		immediato			assoluto			pag zero			accum			implicito			(ind,X)			(ind),Y			pag 0,X			assol,X			assol,Y			relativo			indiretto			pag 0,Y			FLAG					
mnemo	operazione	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	N	Z	C	I	D	V
LDX	X←M *	A2	2	2	AE	4	3	A6	3	2																			BE	4	3							B6	4	2	X	X				
LDY	Y←M *	A0	2	2	AC	4	3	A4	3	2													B4	4	2	BC	4	3													X	X				
LSR	0→[7 0]→C				4E	6	3	46	5	2	4A	2	1										56	6	2	5E	7	3													0	X	X			
NOP	nessuno													EA	2	1																														
ORA	A←A∨M	09	2	2	0D	4	3	05	3	2							01	6	2	11	5	2	15	4	2	1D	4	3	19	4	3										X	X				
PHA	SK←A SP←SP-1													48	3	1																														
PHP	SK←P SP←SP-1													08	3	1																														
PLA	A←SK SP←SP+1													68	4	1																									X	X				
PLP	P←SK SP←SP+1													28	4	1																									X	X	X	X	X	X
ROL	C←[7 0]←C				2E	6	3	26	5	2	2A	2	1										36	6	2	3E	7	3													X	X	X			
ROR	C→[7 0]→C				6E	6	3	66	5	2	6A	2	1										76	6	2	7E	7	3													X	X	X			
RTI	VEDI CAP 5.4													40	6	1																									X	X	X	X	X	X
RTS	VEDI CAP 5.4													60	6	1																														
SBC	A←A-M-C̄ *	E9	2	2	ED	4	3	E5	3	2							E1	6	2	F1	5	2	F5	4	2	FD	4	3	F9	4	3										X	X	X			X
SEC	C←1													38	2	1																											1			
SED	D←1													F8	2	1																													1	
SEI	I←1													78	2	1																												1		
STA	M←A				8D	4	3	85	3	2							81	6	2	91	6	2	95	4	2	9D	5	3	99	5	3															
STX	M←X				8E	4	3	86	3	2																												96	4	2						
STY	M←Y				8C	4	3	84	3	2													94	4	2																					
TAX	X←A													AA	2	1																									X	X				
TAY	Y←A													A8	2	1																									X	X				
TSX	X←S													BA	2	1																									X	X				
TXA	A←X													8A	2	1																									X	X				
TXS	S←X													9A	2	1																														
TYA	A←Y													98	2	1																									X	X				

* SE A CAVALLO DI PAGINA N=N+1
** SE VERIFICATO N=N+1
SE VERIFICATO E SALTA OLTRE LA PAGINA N=N+2
*** IN MODO DECIMALE IL FLAG DI ZERO NON E' VALIDO

X=REGISTRO X
Y=REGISTRO Y
A=ACCUMULATORE
M=OPERANDO
SK=CELLA DELLO STACK
SP=STACK POINTER

\+ SOMMA
\- SOTTRAZIONE
∧ AND LOGICO
∨ OR LOGICO
⊻ OR ESCLUSIVO

X MODIFICA
M7 BIT 7 DELLA CELLA CONSIDERATA
M6 BIT 6 DELLA CELLA CONSIDERATA
N NUMERO DI CICLI DI CLOCK
\# NUMERO DI BYTES

CODICI E NUMERI DEL CALCOLATORE

La memoria del calcolatore e' formata da celle che vengono chiamate BYTE. Ogni byte e' contraddistinto da un INDIRIZZO numerico che puo' variare da 0 a 65535, come indicato in Figura C.1.

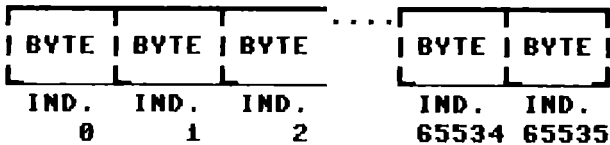



Figura C.1 Indirizzi di memoria

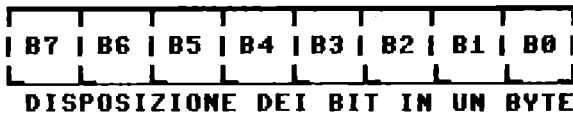
Ogni BYTE e' formato da 8 BIT, cifre binarie che possono essere 0 o 1, come schematizzato in Figura C.2.

NOTA: BI SIGNIFICA BIT DI POSIZIONE I
B0 E' IL BIT MENO SIGNIFICATIVO
B7 E' IL BIT PIU' SIGNIFICATIVO

Figura C.2 Disposizione dei BIT in un BYTE

In ogni BYTE si puo' rappresentare un numero binario formato da 8 BIT (0 o 1); i pesi di ogni cifra sono quelli indicati in Figura C.3.

Figura C.3 Pesi dei BIT in un BYTE

Per calcolare il valore decimale N del numero contenuto in un byte si deve usare la formula che segue, dove "bi" significa "bit di posizione i" e "2^i" signica "2 elevato a i":

N=b7*2^7+b6*2^6+b5*2^5+b4*2^4+b3*2^3+b2*2^2+b1*2^1+b0*2^0
cioe':
N=b7*128+b6*64+b5*32+b4*16+b3*8+b2*4+b1*2+b0*1.

Si ottiene per N il valore 0 quando tutti i bit sono 0, e il valore 255 quando tutti i bit sono 1, cioe' si possono ottenere 256 valori diversi.

L'interpretazione che viene data al numero contenuto in un byte dipende dal contesto nel quale esso viene preso in esame. Durante la lista (LIST) di un programma il numero contenuto in un byte puo' dar luogo a una parola chiave del linguaggio BASIC. Uno o piu' byte contigui possono rappresentare un numero reale espresso in forma esponenziale (floating-point) o un numero intero. Una serie contigua di byte puo' rappresentare una parola codificata carattere per carattere.

I CODICI

Per rappresentare i caratteri nella memoria del calcolatore o nei supporti magnetici di memorizzazione, come dischi o nastri, o per

inviare dati alla stampante il sistema usa il codice CBM-ASCII, occupando un byte per ogni carattere. In questo codice sono disponibili 256 caratteri diversi, codificati da 0 a 255. Non tutti questi caratteri sono stampabili; alcuni sono usati come codici di controllo per produrre effetti particolari.
Il COMMODORE PLUS-4 dispone di due gruppi diversi di codici ASCI-I, si dice di due SET di caratteri; il primo si chiama SET MAIUSCOLO/GRAFICO e il secondo SET MINUSCOLO/MAIUSCOLO. E' possibile usare i due SET di caratteri in alternativa, mai contemporaneamente.
Per evidenziare i caratteri sul video (cioe' per scrivere o disegnare) il sistema usa un codice leggermente diverso dal codice ASCII, ma con esso relazionato, che si chiama D/CODE.
Analizziamo i modi che l'utente ha per inviare dati da tastiera al calcolatore:
.1) Il calcolatore e' in stato comandi, si possono scrivere comandi in immediato o istruzioni di programma, le parole chiave del linguaggio e i simboli sono trasformati in TOKENS, come indicato in Tabella C.2; le costanti numeriche o stringa, i nomi delle variabili e delle funzioni utente sono memorizzate in codice ASCII, come indicato in Tabella C.1.
.2) Il calcolatore e' in stato programma e esegue una istruzione di richiesta dati da tastiera; esso accetta caratteri codificati ASCII, che corrispondono ai tasti premuti, e li trasforma nel formato richiesto dal tipo di variabile alla quale sono destinati, con eventuale segnalazione di errore. Gli errori possibili sono:
..variabile numerica contro dato alfabetico,
..variabile stringa contro dato piu' lungo di 255 caratteri.
In ambedue gli stati indicati il sistema fa comparire quello che si scrive, carattere per carattere, sul video nella posizione attuale del cursore.
Riepiloghiamo i modi che ha l'utente per scrivere da programma sul video:
.1) Usare l'istruzione PRINT seguita dal nome di una o piu' variabili e/o da una o piu' costanti separate dai separatori consentiti. Il sistema fa comparire i dati carattere per carattere a partire dalla posizione attuale del cursore; eventuali caratteri di controllo presenti possono spostare la posizione del cursore. I caratteri appaiono del colore attivo per il testo. Usando i codici di controllo del colore si puo' modificare il colore dei caratteri.
La lista dei dati da stampare, che segue la parola chiave PRINT, puo' comprendere anche funzioni, come, per esempio, la CHR$. Se l'argomento di CHR$ e' il codice di un carattere stampabile, esso compare, se, invece, esso e' un codice di controllo, produce il suo effetto.
.2) Usare l'istruzione CHAR che serve per scrivere una stringa di caratteri in una definita posizione del video.

.3) Usare due istruzioni POKE per memorizzare in una posizione della MAPPA VIDEO il D/CODE di un carattere e nella corrispondente posizione della MAPPA COLORE il codice del colore desiderato, che deve essere diverso dal colore dello sfondo. In modo testo al video corrisponde una zona di memoria (MAPPA VIDEO) che puo' contenere i codici (D/CODE) dei 1000 caratteri visualizzabili, e una zona di memoria (MAPPA COLORE) per i codici del colore che deve avere ogni carattere.

Nei casi 1 e 2 e' il sistema che esegue le due POKE necessarie per evidenziare un carattere sul video.

I caratteri stampabili di ogni SET sono 128; essi pero' possono essere evidenziati in due modi, positivo e negativo (si dice in campo diretto e in campo inverso, cioe' scambiando tra loro il colore dello sfondo e il colore del carattere). I D/CODE dei caratteri stampabili diretti vanno da 0 a 127, quelli dei caratteri inversi, da 128 a 255.

Nella Tabella C.1 sono riportati tutti i caratteri stampabili, sia diretti che inversi, per i due SET di caratteri disponibili, i codici ASCII, espressi in decimale, e i D/CODE dei caratteri diretti e dei caratteri inversi, espressi in decimale. Usando la funzione CHR$ con argomento uguale al codice decimale ASCII si ottiene la stampa del carattere diretto. Per ottenere il carattere inverso si deve usare il codice di controllo necessario per attivare RVS-ON (CHR$(18)). Quando invece si scrive con l'istruzione POKE si puo' usare il D/CODE del carattere inverso.

CORRISP. CARATTERI, CODICI ASCII, D/CODE PER I DUE SET DISPONIBILI

MA/GRAF		MIN/MA		ASCII	D/CODE	
DIR.	INV.	DIR.	INV.	DEC.	DIR.	INV.
	■		■	32	32	160
!	!	!	!	33	33	161
"	"	"	"	34	34	162
#	#	#	#	35	35	163
$	$	$	$	36	36	164
%	%	%	%	37	37	165
&	&	&	&	38	38	166

TABELLA C.1 Caratteri, codice ASCII e D-CODE

CORRISP. CARATTERI, CODICI ASCII, D/CODE
PER I DUE SET DISPONIBILI

MA/GRAF DIR.	MA/GRAF INV.	MIN/MA DIR.	MIN/MA INV.	ASCII DEC.	D/CODE DIR.	D/CODE INV.
'	'	'	'	39	39	167
(	(	(	(	40	40	168
)	)	)	)	41	41	169
*	*	*	*	42	42	170
+	+	+	+	43	43	171
,	,	,	,	44	44	172
−	−	−	−	45	45	173
.	.	.	.	46	46	174
/	/	/	/	47	47	175
0	0	0	0	48	48	176
1	1	1	1	49	49	177
2	2	2	2	50	50	178
3	3	3	3	51	51	179
4	4	4	4	52	52	180
5	5	5	5	53	53	181
6	6	6	6	54	54	182
7	7	7	7	55	55	183
8	8	8	8	56	56	184
9	9	9	9	57	57	185
:	:	:	:	58	58	186
;	;	;	;	59	59	187
<	<	<	<	60	60	188
=	=	=	=	61	61	189
>	>	>	>	62	62	190
?	?	?	?	63	63	191
@	@	@	@	64	0	128
A	A	a	a	65	1	129
B	B	b	b	66	2	130
C	C	c	c	67	3	131
D	D	d	d	68	4	132
E	E	e	e	69	5	133

TABELLA C.1 Caratteri, codice ASCII e D-CODE (continuazione)

CORRISP. CARATTERI, CODICI ASCII, D/CODE PER I DUE SET DISPONIBILI

MA/GRAF DIR.	MA/GRAF INV.	MIN/MA DIR.	MIN/MA INV.	ASCII DEC.	D/CODE DIR.	D/CODE INV.
F	F	f	f	70	6	134
G	G	g	g	71	7	135
H	H	h	h	72	8	136
I	I	i	i	73	9	137
J	J	j	j	74	10	138
K	K	k	k	75	11	139
L	L	l	l	76	12	140
M	M	m	m	77	13	141
N	N	n	n	78	14	142
O	O	o	o	79	15	143
P	P	p	p	80	16	144
Q	Q	q	q	81	17	145
R	R	r	r	82	18	146
S	S	s	s	83	19	147
T	T	t	t	84	20	148
U	U	u	u	85	21	149
V	V	v	v	86	22	150
W	W	w	w	87	23	151
X	X	x	x	88	24	152
Y	Y	y	y	89	25	153
Z	Z	z	z	90	26	154
[	[	[	[	91	27	155
£	£	£	£	92	28	156
]	]	]	]	93	29	157
↑	↑	↑	↑	94	30	158
←	←	←	←	95	31	159
—	—	—	—	96	64	192
♠	♠	A	A	97	65	193
|	|	B	B	98	66	194
—	—	C	C	99	67	195
—	—	D	D	100	68	196
—	—	E	E	101	69	197
—	—	F	F	102	70	198
|	|	G	G	103	71	199

TABELLA C.1 Caratteri, codice ASCII e D-CODE (continuazione)

CORRISP. CARATTERI, CODICI ASCII, D/CODE
PER I DUE SET DISPONIBILI

MA/GRAF DIR.	MA/GRAF INV.	MIN/MA DIR.	MIN/MA INV.	ASCII DEC.	D/CODE DIR.	D/CODE INV.
		H		104	72	200
		I		105	73	201
		J		106	74	202
		K		107	75	203
L		L		108	76	204
\		M		109	77	205
/		N		110	78	206
		O		111	79	207
		P		112	80	208
●		Q		113	81	209
		R		114	82	210
♥		S		115	83	211
		T		116	84	212
		U		117	85	213
X		V		118	86	214
		W		119	87	215
		X		120	88	216
		Y		121	89	217
◆		Z		122	90	218
+		+		123	91	219
				124	92	220
				125	93	221
				126	94	222
				127	95	223
				160	96	224
				161	97	225
				162	98	226
				163	99	227
				164	100	228
				165	101	229
				166	102	230
				167	103	231

TABELLA C.1 Caratteri, codice ASCII e D-CODE (continuazione)

CORRISP. CARATTERI, CODICI ASCII, D/CODE
PER I DUE SET DISPONIBILI

MA/GRAF		MIN/MA		ASCII	D/CODE	
DIR.	INV.	DIR.	INV.	DEC.	DIR.	INV.
				168	104	232
				169	105	233
				170	106	234
				171	107	235
				172	108	236
				173	109	237
				174	110	238
				175	111	239
				176	112	240
				177	113	241
				178	114	242
				179	115	243
				180	116	244
				181	117	245
				182	118	246
				183	119	247
				184	120	248
				185	121	249
		✓		186	122	250
				187	123	251
				188	124	252
				189	125	253
				190	126	254
				191	127	255
				192	64	192
♠		A	A	193	65	193
		B	B	194	66	194
		C	C	195	67	195
		D	D	196	68	196
		E	E	197	69	197
		F	F	198	70	198
		G	G	199	71	199
		H	H	200	72	200

TABELLA C.1 Caratteri, codice ASCII e D-CODE (continuazione)

CORRISP. CARATTERI, CODICI ASCII, D/CODE PER I DUE SET DISPONIBILI

MA/GRAF DIR.	MA/GRAF INV.	MIN/MA DIR.	MIN/MA INV.	ASCII DEC.	D/CODE DIR.	D/CODE INV.
		I		201	73	201
		J		202	74	202
		K		203	75	203
		L		204	76	204
		M		205	77	205
		N		206	78	206
		O		207	79	207
		P		208	80	208
		Q		209	81	209
		R		210	82	210
		S		211	83	211
		T		212	84	212
		U		213	85	213
		V		214	86	214
		W		215	87	215
		X		216	88	216
		Y		217	89	217
		Z		218	90	218
+		+		219	91	219
				220	92	220
				221	93	221
				222	94	222
				223	95	223
				224	96	224
				225	97	225
				226	98	226
				227	99	227
				228	100	228
				229	101	229
				230	102	230
				231	103	231
				232	104	232
				233	105	233
				234	106	234

TABELLA C.1 Caratteri, codice ASCII e D-CODE (continuazione)

CORRISP. CARATTERI, CODICI ASCII, D/CODE PER I DUE SET DISPONIBILI

MA/GRAF DIR.	MA/GRAF INV.	MIN/MA DIR.	MIN/MA INV.	ASCII DEC.	D/CODE DIR.	D/CODE INV.
├	▌:	├	▌:	235	107	235
▪	▛	▪	▛	236	108	236
└	▙	└	▙	237	109	237
┐	▜	┐	▜	238	110	238
_	▀	_	▀	239	111	239
┌	▛	┌	▛	240	112	240
┴	▄	┴	▄	241	113	241
┬	▀	┬	▀	242	114	242
┤	▐	┤	▐	243	115	243
▏	▐	▏	▐	244	116	244
▎	▐	▎	▐	245	117	245
▕	▌	▕	▌	246	118	246
▔	▄	▔	▄	247	119	247
▀	▄	▀	▄	248	120	248
▁	▀	▁	▀	249	121	249
┘	▀	✓	▨	250	122	250
▖	▜	▖	▜	251	123	251
▝	▙	▝	▙	252	124	252
▘	▟	▘	▟	253	125	253
▗	▟	▗	▟	254	126	254
π	▣	※	※	255	94	222

TABELLA C.1 Caratteri, codice ASCII e D-CODE (continuazione)

La Tabella C.1 si riferisce ai soli caratteri stampabili, che sono 128 diretti e 128 inversi; le prime due colonne riportano i caratteri del SET MAIUSCOLO /GRAFICO, le due colonne successive quelli del SET MINUSCOLO/MAIUSCOLO.
I codici ASCII vanno da 32 a 127 e da 160 a 255; in realta' i codici da 192 a 223 producono gli stessi caratteri dei codici da 96 a 127, i codici da 224 a 254 producono gli stessi caratteri dei codici da 160 a 190, e il codice 255 produce lo stesso carattere del codice 126.

La Tabella C.2 riguarda i TOKENS, cioe' i codici corrispondenti alle parole chiave e ai simboli del BASIC. In essa la prima

colonna riporta il codice, la seconda la parola chiave o il simbolo corrispondente, la terza il modo, se esiste, per abbreviare la scrittura, la quarta l'effetto prodotto sul video dall'abbreviazione. Nella Tabella C.2 e' stato seguito quasi l'ordine alfabetico delle parole chiave, ponendo in fondo i simboli di tipo aritmetico. Le eccezioni all'ordine alfabetico delle parole chiave si hanno per:

..FN, che segue DEF, dato che si usano insieme,
..THEN e ELSE che seguono IF, dato che sono usate insieme a IF,
..USING che segue PRINT e PRINT#, dato che viene usato insieme ad esse.

T O K E N S

CODICE	KEYWORD	ABBREVIAZIONE	VISUALIZZAZIONE
182	ABS	A SHIFT-B	A│
175	AND	A SHIFT-N	A╱
198	ASC	A SHIFT-S	A♥
193	ATN	A SHIFT-T	A│
220	AUTO	A SHIFT-U	A╭
246	BACKUP	B SHIFT-A	B♠
225	BOX	B SHIFT-O	B┌
224	CHAR	CH SHIFT-A	CH♠
199	CHR$	C SHIFT-H	C│
226	CIRCLE	C SHIFT-I	C╮
160	CLOSE	CL SHIFT-O	CL┌
156	CLR	C SHIFT-L	C└
157	CMD	C SHIFT-M	C╲
243	COLLECT	COL SHIFT-L	COL└

TABELLA C.2 Parole chiave, TOKENS e abbreviazioni

T O K E N S

CODICE	KEYWORD	ABBREVIAZIONE	VISUALIZZAZIONE
231	COLOR	CO SHIFT-L	CO└
154	CONT	C SHIFT-O	C┌
244	COPY	CO SHIFT-P	CO┐
190	COS	MANCA	MANCA
131	DATA	D SHIFT-A	D♠
209	DEC	MANCA	MANCA
150	DEF	D SHIFT-E	D▔
165	FN	MANCA	MANCA
247	DELETE	DE SHIFT-L	DE└
134	DIM	D SHIFT-I	D╮
238	DIRECTORY	DI SHIFT-R	DI▁
240	DLOAD	D SHIFT-L	D└
235	DO	MANCA	MANCA
229	DRAW	D SHIFT-R	D▁
239	DSAVE	D SHIFT-S	D♥
128	END	E SHIFT-N	E╱
211	ERR$	E SHIFT-R	E▁
237	EXIT	EX SHIFT-I	EX╮
189	EXP	E SHIFT-X	E♣
129	FOR	F SHIFT-O	F┌
184	FRE	F SHIFT-R	F▁
161	GET	G SHIFT-E	G▔
203	GO	MANCA	MANCA
141	GOSUB	GO SHIFT-S	GO♥
137	GOTO	G SHIFT-O	G┌
222	GRAPHIC	G SHIFT-R	G▁
227	GSHAPE	G SHIFT-S	G♥
241	HEADER	HE SHIFT-A	HE♠
234	HELP	HE SHIFT-L	HE└
210	HEX$	H SHIFT-E	H▔
139	IF	MANCA	MANCA
167	THEN	T SHIFT-H	T│
213	ELSE	E SHIFT-L	E└
133	INPUT	MANCA	MANCA
132	INPUT#	I SHIFT-N	I╱
212	INSTR	IN SHIFT-S	IN♥

TABELLA C.2 Parole chiave, TOKENS e abbreviazioni (continuazione)

T O K E N S

CODICE	KEYWORD	ABBREVIAZIONE	VISUALIZZAZIONE
181	INT	MANCA	MANCA
207	JOY	J SHIFT-O	J┏
249	KEY	K SHIFT-E	K‾
200	LEFT$	LE SHIFT-F	LE▁
195	LEN	MANCA	MANCA
136	LET	L SHIFT-E	L‾
155	LIST	L SHIFT-I	L╲
147	LOAD	L SHIFT-O	L┏
230	LOCATE	LO SHIFT-C	LO─
188	LOG	MANCA	MANCA
236	LOOP	LO SHIFT-O	LO┏
202	MID$	M SHIFT-I	M╲
250	MONITOR	M SHIFT-O	M┏
162	NEW	MANCA	MANCA
130	NEXT	N SHIFT-E	N‾
168	NOT	N SHIFT-O	N┏
145	ON	MANCA	MANCA
159	OPEN	O SHIFT-P	O┓
176	OR	MANCA	MANCA
223	PAINT	P SHIFT-A	P♠
194	PEEK	P SHIFT-E	P‾
151	POKE	P SHIFT-O	P┏
185	POS	MANCA	MANCA
153	PRINT	?	?
152	PRINT#	P SHIFT-R	P▁
251	USING	US SHIFT-I	US╲
221	PUDEF	P SHIFT-U	P╭
205	RCLR	R SHIFT-C	R─
208	RDOT	R SHIFT-D	R─
135	READ	R SHIFT-E	R‾
143	REM	MANCA	MANCA
245	RENAME	RE SHIFT-N	RE╱
248	RENUMBER	REN SHIFT-U	REN╭
140	RESTORE	RE SHIFT-S	RE♥
214	RESUME	RES SHIFT-U	RES╭
142	RETURN	RE SHIFT-T	RE|
204	RGR	R SHIFT-G	R|
201	RIGHT$	R SHIFT-I	R╲

TABELLA C.2 Parole chiave, TOKENS e abbreviazioni (continuazione)

T O K E N S

CODICE	KEYWORD	ABBREVIAZIONE	VISUALIZZAZIONE
206	RLUM	R SHIFT-L	R└
187	RND	R SHIFT-N	R╱
138	RUN	R SHIFT-U	R╭
148	SAVE	S SHIFT-A	S♠
233	SCALE	SC SHIFT-A	SC♠
232	SCNCLR	S SHIFT-C	S─
242	SCRATCH	SC SHIFT-R	SC▁
180	SGN	S SHIFT-G	S│
191	SIN	S SHIFT-I	S╮
218	SOUND	S SHIFT-O	S┌
166	SPC(	S SHIFT-P	S┐
186	SQR	S SHIFT-Q	S●
228	SSHAPE	S SHIFT-S	S♥
169	STEP	ST SHIFT-E	ST▔
144	STOP	S SHIFT-T	S│
196	STR$	ST SHIFT-R	ST▁
158	SYS	S SHIFT-Y	S │
163	TAB(	T SHIFT-A	T♠
192	TAN	MANCA	MANCA
164	TO	MANCA	MANCA
215	TRAP	T SHIFT-R	T▁
217	TROFF	TRO SHIFT-F	TRO▁
216	TRON	TR SHIFT-O	TR┌
252	UNTIL	U SHIFT-N	U╱
183	USR	U SHIFT-S	U♥
197	VAL	V SHIFT-A	V♠
149	VERIFY	V SHIFT-E	V▔
219	VOL	V SHIFT-O	V┌
146	WAIT	W SHIFT-A	W♠
253	WHILE	W SHIFT-H	W│
170	+	+	+
171	−	−	−
172	*	*	*
173	/	/	/
174	↑	↑	↑
177	>	>	>
178	=	=	=
179	<	<	<

TABELLA C.2 Parole chiave, TOKENS e abbreviazioni (continuazione)

Nella Tabella C.3 riportiamo l'elenco dei TOKENS in ordine di codice crescente.

TOKENS ORDINATI

CODICE	KEYWORD	CODICE	KEYWORD
128	END	160	CLOSE
129	FOR	161	GET
130	NEXT	162	NEW
131	DATA	163	TAB(
132	INPUT#	164	TO
133	INPUT	165	FN
134	DIM	166	SPC(
135	READ	167	THEN
136	LET	168	NOT
137	GOTO	169	STEP
138	RUN	170	+
139	IF	171	-
140	RESTORE	172	*
141	GOSUB	173	/
142	RETURN	174	↑
143	REM	175	AND
144	STOP	176	OR
145	ON	177	>
146	WAIT	178	=
147	LOAD	179	<
148	SAVE	180	SGN
149	VERIFY	181	INT
150	DEF	182	ABS
151	POKE	183	USR
152	PRINT#	184	FRE
153	PRINT	185	POS
154	CONT	186	SQR
155	LIST	187	RND
156	CLR	188	LOG
157	CMD	189	EXP
158	SYS	190	COS
159	OPEN	191	SIN

192	TAN	214	RESUME
193	ATN	215	TRAP
194	PEEK	216	TRON
195	LEN	217	TROFF
196	STR$	218	SOUND
197	VAL	219	VOL
198	ASC	220	AUTO
199	CHR$	221	PUDEF
200	LEFT$	222	GRAPHIC
201	RIGHT$	223	PAINT
202	MID$	224	CHAR
203	GO	225	BOX
204	RGR	226	CIRCLE
205	RCLR	227	GSHAPE
206	RLUM	228	SSHAPE
207	JOY	229	DRAW
208	RDOT	230	LOCATE
209	DEC	231	COLOR
210	HEX$	232	SCNCLR
211	ERR$	233	SCALE
212	INSTR	234	HELP
213	ELSE	235	DO

TABELLA C.3 TOKENS in ordine di codice

Dalla Tabella C.2 sono escluse le parole riservate riportate nella Tabella C.4; esse sono codificate carattere per carattere in codice ASCII. Inoltre, l'istruzione GET# viene memorizzata utilizzanto il codice di GET, seguito dal codice ASCII di #, cioe' come 161 e 35 in due byte.
Per il significato delle variabili della Tabella C.4, rimandiamo all'Appendice A.

CODICE	KEYWORD		
236	LOOP	242	SCRATCH
237	EXIT	243	COLLECT
238	DIRECTORY	244	COPY
239	DSAVE	245	RENAME
240	DLOAD	246	BACKUP
241	HEADER	247	DELETE

248	RENUMBER	251	USING
249	KEY	252	UNTIL
250	MONITOR	253	WHILE

Nella Tabella C.5 riportiamo il significato di alcuni dei codici ASCII che vanno da 0 a 31 e da 128 a 160, quando sono usati come argomento della funzione CHR$. Per alcuni codici viene indicato anche l'effetto prodotto relativamente alla stampante MPS803.

PAROLE RISERVATE CODIFICATE IN ASCII	
NOME	CODIFICA ASCII
DS	68 83
DS$	68 83 36
EL	69 76
ER	69 82
ERR$	69 82 82 36
ST	83 84
TI	84 73
TI$	84 73 36
π	255

TABELLA C.4 Parole riservate codificate ASCII

C O D I C I   D I   C O N T R O L L O

CODICI	SIGNIFICATO
5	COLORE BIANCO (CTRL-2)
8	DISABILITA SHIFT-CBM
	MPS801 ATTIVA MODO GRAFICO PER PUNTI
9	ABILITA SHIFT-CBM
10	MPS801 MANDA LINE FEED
13	RETURN
	MPS801 RETURN + LINE FEED
14	ATTIVA SET MAIUSC./MINUSC.
	MPS801 ATTIVA DOPPIA AMPIEZZA
15	MPS801 DISABILITA COD. 14
16	MPS801 SPOSTAMENTO POS.
17	EFFETTO DI FRECCIA GIU'
	MPS801 ATTIVA SET MINUSC./MAIUSC.
18	ATTIVA RVS ON
	MPS801 ATTIVA RVS ON
19	CURSORE IN POSIZIONE HOME
20	CANCELLA (DELETE)
26	MPS801 RIPETE CAR. GRAFICI
27	FUNZIONE DI ESCAPE
	MPS801 SPOSTA A UNA POSIZIONE PUNTO
28	COLORE ROSSO (CTRL-3)
29	EFFETTO FRECCIA A DESTRA
30	COLORE VERDE (CTRL-6)
31	COLORE BLU (CTRL-7)
129	COLORE ARANCIONE (CBM-1)
130	FLASH ON
131	FLASH OFF

TABELLA C.5 Codici di controllo

CODICI	SIGNIFICATO
141	SHIFT-RETURN
142	ATTIVA SET MAIUSC./GRAF.
144	COLORE NERO (CTRL-1)
145	EFFETTO FRECCIA SU MPS801 ATTIVA SET MAIUSC./GRAF.
146	DISATTIVA RVS ON (RVS OFF)
147	PULIZIA VIDEO E CURSORE IN POS. HOME
148	ATTIVA INSERIMENTO (INS)
149	COLORE BRUNO (CBM-2)
150	COL. GIALLO VERDE (CBM-3)
151	COLORE ROSA (CBM-4)
152	COLORE BLU VERDE (CBM-5)
153	COLORE BLU CHIARO (CBM-6)
154	COLORE BLU SCURO (CBM-7)
155	COL. VERDE CHIARO (CBM-8)
156	COLORE PORPORA (CTRL-5)
157	EFFETTO FRECCIA A SINISTRA
158	COLORE GIALLO (CTRL-8)
159	COLORE CIANO (CTRL-4)

TABELLA C.5 Codici di controllo (continuazione)

Nella Tabella C.6 sono indicate le relazioni tra i codici ASCII.

RELAZIONE TRA CODICE ASCII E D-CODE	
ASCII	**D-CODE**
32<=A<= 63	D = A
64<=A<= 95	D = A - 64
96<=A<=127	D = A - 32
160<=A<=191	D = A - 64
192<=A<=254	D = A -128
A =255	D = A -161

TABELLA C.6 Relazione tra codice ASCII e D-CODE

Riepiloghiamo i modi disponibili per cambiare il set di caratteri.

.1) Premere contemporaneamente i tasti SHIFT e CBM fa passare dal set attivo all'altro. La pressione di questi due tasti non ha effetto se e' stato prima eseguito il comando: PRINT CHR$(8), che disabilita i tasti SHIFT-CBM. Nel caso basta eseguire il comando: PRINT CHR$(9) per abilitare nuovamente i tasti SHIFT-CBM.

.2) Eseguire il comando: PRINT CHR$(142) per attivare il set maiuscolo/grafico. Eseguire il comando PRINT CHR$(14) per attivare il set minuscolo/maiuscolo.

I caratteri, che vengono evidenziati sul video, sono descritti per punti in una zona di memoria ROM del calcolatore. Ogni carattere e' rappresentato da 64 punti disposti in una matrice di 8

righe e 8 colonne; in conseguenza per descrivere un carattere vengono usati 8 byte, cioe' 64 bit. Il primo byte rappresenta i punti della prima riga, il secondo quelli della seconda riga, e cosi' via. I bit a 1 corrispondono ai punti da evidenziare nel colore del testo (inchiostro), i bit a 0 ai punti dove lasciare invariato il colore dello sfondo. Quando un carattere deve essere visualizzato sul video, il sistema, usando come puntatore il D/CODE del carattere, va a prelevare la sua descrizione dalla tabella relativa e lo disegna per punti nella posizione attuale del cursore.

```
***STAMPA IMMAGINE CARATTERI***

D/CODE= 0  SET MAIUSCOLO/GRAFICO

  CARATTERE       BINARIO      DECIMALE

    ****          00111100       60
   **  **         01100110       102
   ** ***         01101110       110
   ** ***         01101110       110
   **             01100000       96
   **   *         01100010       98
    ****          00111100       60
                  00000000       0
***STAMPA IMMAGINE CARATTERI***

D/CODE= 1  SET MAIUSCOLO/GRAFICO

  CARATTERE       BINARIO      DECIMALE

     **           00011000       24
    ****          00111100       60
   **  **         01100110       102
   ******         01111110       126
   **  **         01100110       102
   **  **         01100110       102
   **  **         01100110       102
                  00000000       0
```

Figura C.4 Descrizione dei caratteri

```
***STAMPA IMMAGINE CARATTERI***

D/CODE= 65  SET MAIUSCOLO/GRAFICO


   CARATTERE          BINARIO        DECIMALE

       *              00001000         8
      ***             00011100         28
     *****            00111110         62
    *******           01111111         127
    *******           01111111         127
      ***             00011100         28
     *****            00111110         62
                      00000000         0
***STAMPA IMMAGINE CARATTERI***

D/CODE= 88  SET MAIUSCOLO/GRAFICO


   CARATTERE          BINARIO        DECIMALE

      **              00011000         24
      **              00011000         24
    **  **            01100110         102
    **  **            01100110         102
      **              00011000         24
      **              00011000         24
     ****             00111100         60
                      00000000         0
```

Figura C.4 Descrizione dei caratteri (continuazione)

Nella Figura C.4 sono riportati i caratteri corrispondenti ai D/CODE 0, 1 65 e 88 nel set maiuscolo/grafico. Essi sono disegnati ingranditi usando un asterisco in corrispondenza di ogni bit a 1. Inoltre, viene riportato il contenuto binario e il valore decimale degli 8 byte che descrivono il carattere.

Quando si considera un numero contenuto in due byte consecutivi si possono presentare i seguenti casi:
..1) I due byte rappresentano un indirizzo usato dal sistema per la gestione del calcolatore. In questo caso il byte di indirizzo minore (il primo) e' il byte basso (LO) e quello successivo (il secondo) e' il byte alto (HI). Per calcolare il numero rappresentato, se N e' l'indirizzo del primo byte, si procede cosi':

Numero = 256*PEEK(N+1) + PEEK(N)

..2) I due byte rappresentano un numero intero con segno. In questo caso il primo byte contiene la parte piu' significativa del numero e il secondo la parte meno significativa. Il bit piu' significativo del primo byte (bit 7) e' usato per il segno; 0 per numeri positivi e 1 per numeri negativi.
Il valore del numero intero positivo puo' variare da 0 a 32767=127*256+255.
Esempi:
numero decimale: 0
contenuto dei due byte binari:
00000000 00000000

numero decimale: 32767
contenuto dei due byte binari:
01111111 11111111

numero decimale: 837
contenuto dei due byte binari:
00000011 01000101

Il valore del numero intero negativo puo' variare da -32768 a -1.

Per arrivare alla rappresentazione del numero negativo (detta in complemento a 2) possiamo procedere cosi':
..consideriamo la potenza del 2 immediatamente superiore al massimo numero rappresentabile (con 15 bit a disposizione, 8 nel byte basso e 7 nel byte alto, si arriva al massimo al valore posizionale di 2^14), che risulta 2^15, cioe' il numero 32768;
..consideriamo il valore assoluto del numero da rappresentare;
..calcoliamo la differenza tra 32768 (2^15) e il valore assoluto del numero;
..scriviamo in binario il risultato del calcolo precedente;
..alla fine aggiungiamo un bit 1 a sinistra per il segno meno.
Esempio: rappresentazione di -837.
Calcoliamo 32768-837=31931 e troviamo la rappresentazione binaria di 31931 usando il metodo delle sottrazioni successive delle potenze di 2:

```
31931 - 16384 = 15547   bit 1 in posizione 14
15547 -  8192 =  7355   bit 1 in posizione 13
 7355 -  4096 =  3259   bit 1 in posizione 12
 3259 -  2048 =  1211   bit 1 in posizione 11
 1211 -  1024 =   187   bit 1 in posizione 10
  187 -   128 =    59   bit 1 in posizione  7
   59 -    32 =    27   bit 1 in posizione  5
   27 ¬    16 =    11   bit 1 in posizione  4
   11 -     8 =     3   bit 1 in posizione  3
    3 -     2 =     1   bit 1 in posizione  1
    1 ¬     1 =     0   bit 1 in posizione  0
```

il contenuto dei due byte risulta, dopo l'aggiunta del bit 1 di segno in posizione 15:

11111100 10111011

e rappresenta ¬837.

Si puo' arrivare allo stesso risultato applicando la regola che segue:

..scrivere la rappresentazione binaria del valore assoluto del numero:

00000011 01000101

..sostituire ai bit 1 dei bit 0 e ai bit 0 dei bit 1, e, alla fine sommare un bit 1 in fondo a destra (1+1=10 e 1+0=1 nel calcolo binario):

con la sostituzione si ha 11111100 10111010
con l'aggiunta si ha 11111100 10111011.
Il numero ottenuto e' la rappresentazione in complemento a 2 del numero negativo.

Il numero -1 ha la rappresentazione in complemento a 2: 11111111 11111111, mentre il numero -32768 ha la rappresentazione in complemento a 2: 10000000 00000000.

Con il programma INTERI, che segue, puoi provare a introdurre numeri interi; vedrai alcune cose interessanti analizzando i risultati.

```
1 REM INTERI
10 INPUT"NUMERO INTERO: ";N%
20 IFN%=0THENSTOP
30 PRINT"STAMPA NUMERO N%: ";N%
40 GOTO10
```

RISULTATI PROGRAMMA INTERI

```
NUMERO INTERO: ? 32767
STAMPA NUMERO N%:  32767
NUMERO INTERO: ?-32767
STAMPA NUMERO N%: -32767
NUMERO INTERO: ? 45.89
STAMPA NUMERO N%:  45
NUMERO INTERO: ? 32768
STAMPA NUMERO N%: -32768
NUMERO INTERO: ? 50000
STAMPA NUMERO N%: -15536
NUMERO INTERO: ?-88567
STAMPA NUMERO N%: -23031
NUMERO INTERO: ?-32768

?ILLEGAL QUANTITY ERROR IN 10
READY.
```

Se rispondi con 32768 viene ricevuto -32768; se rispondi con 50000 viene ricevuto -15536, se rispondi con -88567 viene ricevuto -23031. In conseguenza se un programma chiede un numero intero si deve RIGOROSAMENTE rispondere con un numero compreso tra -32767 e +32767. Come puoi vedere dai risultati del programma INTERI, se introduci il numero -32768 ottieni un messaggio di errore.

Quando un singolo byte rappresenta un numero senza segno, esso varia da 0 a 255. Se si considera il primo bit a sinistra per il segno, allora il numero positivo varia da 0 a 127 e il numero negativo (applicando le regole precedenti al singolo byte) varia da -128 a -1.

I numeri non interi, cioe' i numeri reali, sono rappresentati nella memoria del calcolatore in forma esponenziale (floating-point) occupando 5 byte consecutivi. Il primo dei 5 byte contiene l'esponente (caratteristica del numero) e gli altri 4 contengono la mantissa, cioe' le cifre significative del numero. Sia la caratteristica che la mantissa sono numeri con segno. Esempi di numeri decimali espressi in forma esponenziale sono i seguenti:

123456=0.123456*10^6
123466=1.23456*10^5

123456=12.3456*10^4
123456=0.00123456*10^8
123.895=0.123895*10^3
123.895=1.23895*10^2
0.789=789*10^(-3)

Di norma viene detta forma normalizzata quella nella quale la mantissa comincia con la prima cifra significativa preceduta dal punto decimale.
Nella memoria del calcolatore sia la caratteristica che la mantissa sono espressi in binario con segno. Per la caratteristica il campo di variabilita' risulta tra -128 e +127, il che corrisponde a una variabilita' in decimale da 10^(-39) a 10^(+38).
Per il campo di variabilita' della mantissa dobbiamo considerare il valore di 4 byte associati, con il piu' significativo a sinistra che inizia con il bit di segno. Inoltre, dato che la mantissa viene calcolata in modo che il primo bit di sinistra sia sempre 1 (come se il numero rappresentato fosse del tipo 0.1... e si siano trascurati lo 0 e il punto decimale), il sistema trascura il primo bit, che e' sempre 1, e lo aggiunge solo quando il numero viene utilizzato per calcoli o visualizzazione. Questo modo di procedere consente di ampliare l'intervallo di variabilita' del numero. In conseguenza i limiti di variabilita' per la mantissa risultano da -4294967304 a +4294967304. Contrariamente a quanto avviene per gli interi, la mantissa dei numeri reali viene sempre conservata in valore assoluto e viene aggiunto il primo bit di sinistra a 0 per i positivi e a 1 per i negativi (rappresentazione in grandezza e segno).
I numeri reali vengono stampati con 9 cifre decimali, ma in memoria ne sono conservate 10.

Con il programma REALI, che segue, puoi provare a introdurre numeri qualunque e controllare come vengono stampati.

```
1 REM REALI
10 INPUT"NUMERO REALE: ";N
20 IFN=0THENSTOP
30 PRINT"STAMPA NUMERO N: ";N
40 GOTO10
```

RISULTATI PROGRAMMA REALI

```
NUMERO REALE: ? 9.99999999
STAMPA NUMERO N:  10
NUMERO REALE: ? 999999999
STAMPA NUMERO N:  999999999
NUMERO REALE: ? 1234.56789123
STAMPA NUMERO N:  1234.56789
NUMERO REALE: ? -9898989898
STAMPA NUMERO N: -9.8989899E+09
NUMERO REALE: ? .00000123456789123
STAMPA NUMERO N:  1.23456789E-06
NUMERO REALE: ? 4294967300
STAMPA NUMERO N:  4.2949673E+09
NUMERO REALE: ? 42949673999
STAMPA NUMERO N:  4.2949674E+10
NUMERO REALE: ? -429496736666
STAMPA NUMERO N: -4.29496737E+11
NUMERO REALE: ? 0
STAMPA NUMERO N:  0
```

MESSAGGI DI ERRORE

errore 1: TOO MANY FILES
Questo messaggio appare quando si cerca di aprire l'undicesimo file. Il COMMODORE PLUS-4 infatti puo' tenere aperti al massimo dieci file contemporaneamente.

errore 2: FILE OPEN
Questo messaggio appare quando si vuole aprire un file logico che gia' e' stato aperto.

errore 3: FILE NOT OPEN
Si cerca di comunicare con un file prima di averlo aperto.

errore 4: FILE NOT FOUND
Il file richiesto non e' presente sul dischetto, o lo sportello del drive non e' stato chiuso, oppure il drive e' guasto. Nelle operazioni su cassetta questo messaggio appare dopo che e' stato incontrato sul nastro il segnale "fine nastro"

errore 5: DEVICE NOT PRESENT
La periferica con cui si e' cercato di comunicare non e' collegata, oppure e' spenta, oppure e' guasta.

errore 6: NOT INPUT FILE
Si cerca di ricevere dati da un file che non e' stato aperto in lettura (ad esempio un file su nastro aperto in scrittura).

errore 7: NOT OUTPUT FILE
Si e' cercato di inviare dati a un file aperto in lettura (ad esempio la tastiera).

errore 8: MISSING FILE NAME
Si e' cercato di salvare un programma su disco senza specificarne il nome.

errore 9: ILLEGAL DEVICE NUMBER
Si e' cercato di eseguire operazioni di ingresso-uscita non consentite per la periferica individuata; ad esempio salvare su unita' 3, che e' il video.

errore 10: NEXT WITHOUT FOR
E' stato incontrato NEXT prima di FOR. Fai molta attenzione a chiudere bene tutti i cicli FOR..NEXT: vedi al proposito il paragrafo 2.7

errore 11: SYNTAX ERROR
E' stato introdotto un comando che il calcolatore non riconosce: controlla bene tutte le lettere della frase introdotta: l'errore di sintassi e' dovuto spesso a errori di battuta. Questo errore appare anche quando si cerca di assegnare dei valori alle variabili riservate, come TI, TI$, ST, DS, DS$, ER, EL. Anche l'uso di parole chiave (GO, TO, IF, OR, ON, ecc) come variabili produce SYNTAX ERROR. Esempio: AGO=21 e' sbagliato perche' AGO viene interpretato come A + GO, e GO e' una parola riservata. A volte puo' capitare questo errore in una linea che appare perfetta: puo' capitare che l'interprete abbia interperetato male delle parole-chiave: ad esempio, la linea:
IFSTAND64THENPRINT"FINE FILE":EXIT
produce SYNTAX ERROR, perche' viene interpretata cosi'.
IF S TAN D64 THEN PRINT.....
(TAN e' una parola riservata) Una linea cosi' e' palesemente sbagliata. Perche' questa linea sia interpretata correttamente occorre porre gli spazi nel modo seguente:
IF ST AND 64 THEN ....
Se pensi che il tuo SYNTAX ERROR sia causato da una situazione di questo tipo, puoi per sicurezza separare tutte le parole chiave con spazi.

errore 12: RETURN WITHOUT GOSUB
E' stata incontrata un'istruzione RETURN senza aver incontrato GOSUB: questo errore capita spesso quando ci si dimentica di porre il comando END alla fine del programma principale, e il calcolatore prosegue, andando ad eseguire le subroutine che seguono il programma.

errore 13: OUT OF DATA
Questo errore l'hai trovato sicuramente tutte le volte che hai premuto RETURN quando il cursore si trovava su una linea che iniziava con la scritta "READY.". Il COMMODORE PLUS-4 interpreta questo messaggio come il comando READ Y e cerca nel programma una linea DATA. Se non la trova segnala questo errore. Nel programma invece questo errore significa che le linee DATA sono state lette tutte, e si e' tentato di leggere piu' dati di quanti sono disponibili.

errore 14: ILLEGAL QUANTITY
Una variabile ha superato il limite consentito per l'operazione richiesta. Aiutati con la funzione HELP e chiedi il valore delle variabili che interessano la linea dove e' avvenuto l'errore; eventualmente controlla i limiti consentiti alle variabili nel comando che lampeggia dopo HELP.

errore 15: OVERFLOW
Una variabile ha raggiunto un valore troppo alto o troppo basso. Leggi il Capitolo 1 e l'appendice C per sapere quali sono i valori massimi e minimi delle variabili.

errore 16: OUT OF MEMORY
Non c'e' posto in memoria per eseguire l'operazione richiesta: per rimediare puoi migliorare la compattazione del tuo programma ponendo molti comandi su una sola linea, o, a volte, eseguire l'istruzione GRAPHIC CLR, e non usare la pagina grafica, oppure ancora dividere il programma in pezzettini (se hai il drive) e caricare di volta in volta le routine che ti servono. Questo messaggio puo' apparire anche quando lo STACK (particolare area di lavoro usata dalle routine del sistema) viene riempito, cioe' quando ci sono troppi FOR..NEXT uno dentro l'altro, con dei GOSUB, con dei DO...LOOP, o con delle espressioni matematiche con molti livelli di parentesi. Prova ad eseguire il programma:

```
10 GOSUB 10
```

esso uscira' con il messaggio OUT OF MEMORY: al BASIC infatti sono riservati 128 bytes di memoria per lo STACK, che non possono essere espansi. Per capire se questo messaggio e' stato causato da un riempimento dello STACK, puoi chiedere quanta memoria e' ancora libera, mediante il comando:

```
PRINT FRE(0)
```

Se il numero e' grande, allora e' probabile che la memoria piena sia lo STACK, se e' piccolo allora la memoria piena dovrebbe essere quella utente. Per avere la certezza sulla causa del messaggio puoi usare la funzione HELP, e vedere la natura del comando che ha causato l'errore. Se il comando era un DIM, o un assegnamento di variabile, allora la memoria piena e' quella utente, se invece il comando e' un GOSUB, o un FOR, o un DO, o il calcolo di un'espressione con delle parentesi, allora la memoria piena e' quella riservata allo STACK. In quest'ultimo caso devi provvedere a cambiare la struttura del tuo programma, in modo da occupare meno posizioni nello STACK.

errore 17: UNDEF'D STATEMENT
Hai eseguito GOTO, GOSUB, RUN, RESTORE, TRAP o RESUME con il numero di una linea che non esiste.

errore 18: BAD SUBSCRIPT

L'indice che hai usato per un variabile con indice supera il valore indicato in sede di dimensionamento, oppure e' negativo.

errore 19: REDIM'D ARRAY
Il calcolatore ha incontrato due volte l'istruzione DIM sulla stessa variabile; questo puo' essere dovuto a un effettivo doppio dimensionamento, oppure al fatto che il BASIC dimensiona automaticamente una variabile con indice a 10, quando si fa riferimento a un suo elemento la prima volta. Ad esempio il comando:
PRINT A(4)
fa si' che venga eseguita una DIM A(10) automaticamente, se il vettore A non era ancora stato dimensionato, e cio' produce questo errore se successivamente si incontra un'istruzione DIM. Se si vuole effettivamente cancellare un vettore, e dimensionarlo nuovamente, allora bisogna eseguire prima una CLR, che cancella tutte le variabili, e poi dimensionare nuovamente il vettore. Ma attenzione: l'istruzione CLR cancella tutto lo STACK, quindi non puo' essere eseguita ne' in una subroutine, ne' in un ciclo.

errore 20: DIVISION BY ZERO
Il divisore di una divisione e' stato di valore 0. Per evitare l'arresto del programma devi controllare il divisore, ed evitare la linea con il calcolo se questo vale zero. Utile in certi casi puo' essere l'istruzione TRAP, che ti permette di eseguire una subroutine speciale dopo un errore; cosi' facendo puoi evitare di calcolare a parte il denominatore, rendendo piu' veloce l'esecuzione del programma.

errore 21: ILLEGAL DIRECT
Alcuni comandi non possono essere eseguiti in modo diretto, ma solo in un programma, come INPUT e GET.

errore 22: TYPE MISMATCH
Il tipo di variabile usato non concorda con quello richiesto; assegnare ad una variabile numerica un valore alfabetico produce questo errore.

errore 23: STRING TOO LONG
Le stringhe possono avere la lunghezza massima di 255 caratteri. Questo messaggio appare facilmente col comando SSHAPE, e appare anche quando introduci linee di comando che superano gli 88 caratteri. Tieni presente che conviene non superare gli 80 caratteri in una linea BASIC, in modo che i listati sulla stampante non debbano andare a capo, e le eventuali correzioni siano piu' facili, avendo a disposizione alcuni caratteri.

errore 24: FILE DATA

I dati letti dal file non sono del tipo richiesto (di solito stringhe al posto di numeri).

errore 25: FORMULA TOO COMPLEX
Questo messaggio appare di solito quando il calcolatore e' in "tilt", a causa di POKE o SYS o routine in linguaggio macchina sbagliate.

errore 26: CAN'T CONTINUE
Il programma puo' riprendere dopo uno STOP, un END, o un BREAK ottenuto premendo il tasto RUN/STOP, a patto di non aver ne' introdotto ne' modificato linee BASIC. Se il programma si ferma per un errore, allora il comando CONT provoca sempre questo messaggio di errore.

errore 27: UNDEF'D FUNCTION
Si e' cercato di calcolare il valore di una funzione non definita.

errore 28: VERIFY ERROR
Il programma che si trova in memoria e' diverso da quello che si trova su nastro o disco: questo errore puo' significare che un file e' stato salvato su un supporto labile, o piu' facilmente che il programma in memoria sia stato leggermente cambiato.

errore 29: LOAD ERROR
Un programma che si trova su nastro non si riesce a caricare: prova a pulire le testine del registratore; se l'errore rimane, e sei sicuro che la cassetta e' stata registrata da un registratore in ordine, allora puoi portare il tuo registratore a un servizio assistenza tecnica, e richiedere di allineare le testine.

errore 30: BREAK ERROR
Hai premuto il tasto RUN/STOP durante l'esecuzione di un programma, oppure hai posto un comando STOP nel programma; se non vuoi che questo messaggio appaia, usa il comando END al posto di STOP.

errore 31: CAN'T RESUME
Il comando RESUME puo' essere eseguito solamente in una routine in cui si e' entrati a causa di un errore, con l'istruzione TRAP. Questo errore appare quando RESUME e' stato incontrato senza l'errore.

errore 32: LOOP NOT FOUND
I cicli DO...LOOP devono terminare con un LOOP, altrimenti viene segnalato questo errore al momento di uscire dal loop.

errore 33: LOOP WITHOUT DO
Questo errore e' molto simile a NEXT WITHOUT FOR; l'istruzione LOOP non ha senso se prima non si e' incontrato un DO.

errore 34: DIRECT MODE ONLY
Alcuni comandi si possono eseguire solo in modo diretto, cioe' non da programma; AUTO, RENUMBER, DELETE sono tra questi.

errore 35: NO GRAPHICS AREA
Si e' cercato di eseguire comandi grafici prima che sia stata riservata memoria all'area grafica con il comando GRAPHIC.

errore 36: BAD DISK
Questo errore appare quando si cerca di eseguire operazioni sul disco, e queste non hanno buon esito; richiedi il valore di DS$ per sapere la natura dell'errore.

UTILIZZO DELLA MEMORIA

Il COMMODORE PLUS-4 utilizza la memoria disponibile con la tecnica dei PUNTATORI. Questo significa che in una zona di memoria RAM, chiamata PAGINA ZERO (corrispondente agli indirizzi piu' bassi), sono memorizzati, in byte di indirizzo prefissato, gli indirizzi di inizio o fine di particolari zone di memoria.

Il programma BASIC dell'utente comincia, di norma, all'indirizzo 4097; l'indirizzo di inizio in memoria del programma BASIC si trova memorizzato nei due byte di indirizzo 43 e 44. Gli indirizzi memorizzati in due byte hanno la parte meno significativa nel primo byte e la piu' significativa nel secondo. Per calcolare l'indirizzo di inizio del programma BASIC si deve eseguire l'istruzione:

```
I=256*PEEK(44)+PEEK(43)
```

ottenendo all'accensione I=4097.

Altri indirizzi utili di puntatori sono:

INIZIO VARIABILI E FINE PROGRAMMA BASIC +1: 45 e 46
INIZIO VARIABILI CON INDICE E FINE VARIABILI SINGOLE: 47 e 48
FINE VARIABILI CON INDICE +1: 49 e 50
INDIRIZZO CIMA CORPI STRINGHE: 51 e 52
PUNTATORE CORPO STRINGHE: 53 e 54
INDIRIZZO PIU' ALTO ZONA DEDICATA AL PROGRAMMA UTENTE: 55 e 56.

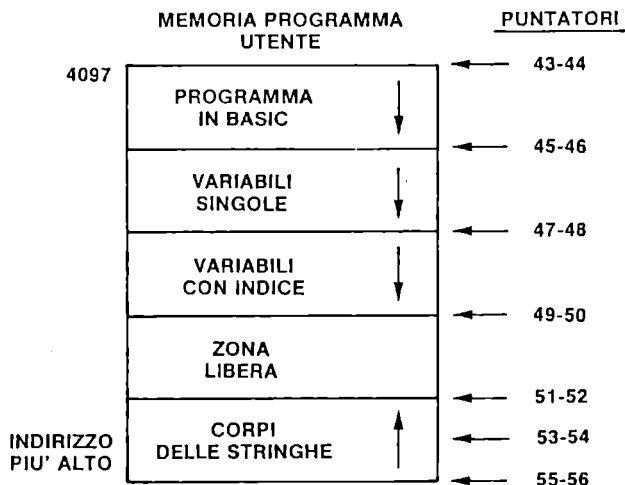



Figura E.1 Utilizzo della memoria e puntatori

Durante la stesura e l'esecuzione di un programma i contenuti di alcuni di questi puntatori cambiano. Se si allunga il programma, le variabili iniziano dopo, se si aggiungono variabili singole si sposta l'inizio delle variabili con indice.

La memoria dedicata al video va da 3072 a 4071. La memoria dedicata ai colori del video va da 2048 a 3047.

Fissiamo la nostra attenzione sulla rappresentazione delle variabili in memoria e utilizziamo alcuni programmi per vedere come esse sono memorizzate.

Iniziamo con il programma INTINMEM. In esso viene chiesto un numero intero e ne viene mostrata la rappresentazione in memoria.

```
1 REM INTINMEM
10 INPUT"{CLR}{DOWN}{DOWN}NUMERO INTERO = ";N%
15 M$="NUMERI INTERI IN MEMORIA"
20 PRINT"{CLR}{DOWN}"M$:PRINT"{DOWN}N% = ";N%
25 A=256*PEEK(46)+PEEK(45)
30 FORK=0TO6:PRINTPEEK(A+K);" ";:NEXTK:PRINT
35 STOP
```

Per vedere cosa c'e' in memoria, viene calcolato l'indirizzo di inizio delle variabili; la variabile N% e' la prima definita nel programma e quindi si trova nei primi byte della zona.

Dai risultati vediamo che per una VARIABILE SINGOLA INTERA sono occupati 7 byte consecutivi; i primi due contengono il nome, e precisamente:
.primo byte: codice ASCII prima lettera del nome + 128, per noi 78, codice di N, +128 da' 206,
.secondo byte: 128 solo, dato che la variabile ha un solo carattere.
La costante aggiuntiva 128 consente di distinguere che si tratta di una variabile con suffisso %.
Ai primi due byte ne seguono 5, dei quali pero' sono utilizzati solo i primi 2, nel modo byte alto che precede il byte basso, come puoi vedere dai risultati. Il programma stampa il contenuto dei 7 byte in decimale (non in binario).

Il programma FLOATINMEM ci permette di vedere come sono memorizzati i numeri reali, cioe' quelli memorizzati in variabili senza suffisso.

```
1 REM FLOATINMEM
10 PRINT"♥QQSCRIVI UN NUMERO DECIMALE":INPUTN
15 IFSW=1THEN25
20 P=PEEK(45)+256*PEEK(46):GOSUB235
25 PRINT"♥QQQ"C1$;N:L$=""
30 IFPEEK(P)<>78THENSTOP
35 FORI=1TO5:X(I)=PEEK(P+I+1):NEXTI
40 PRINTC2$
45 FORI=2TO5:PRINTX(I);"       ";:NEXTI
50 PRINT:PRINTO$
55 FORI=2TO5:GOSUB165:PRINTF$(I);" ";:NEXTI
60 PRINT
65 PRINTC3$:PRINTL$
70 S=1+2*(LEFT$(F$(2),1)="1")
75 F$(2)="1"+RIGHT$(F$(2),7)
80 PRINTN1$:PRINTN2$
85 LL$=F$(2)+F$(3)+F$(4)+F$(5)
90 PRINTLL$
95 Y=X(1)-128
100 PRINTC8$;X(1)
105 PRINTC5$;Y
110 IFY<0THENM$="0":D$=LEFT$(Z$,-Y)+LL$:GOTO140
115 IFX(1)=0THENM$="0":D$="0":GOTO140
120 IFY>LEN(LL$)THENLL$=LEFT$(LL$+Z$,Y)
125 M$=LEFT$(LL$,Y):Z=LEN(LL$)-Y
130 IFZ<=0THEND$="0":GOTO140
135 D$=RIGHT$(LL$,(LEN(LL$)-Y))
140 PRINTC6$,M$:PRINTC7$,D$
145 GOSUB195:GOSUB215:PRINTC4$;(M+D)*S
150 INPUT"QQANCORA (S/N) ";B$
155 IFB$="S"THENSW=1:GOTO10
160 STOP
165 F$(I)="":FORJ=7TO0STEP-1
170 IFINT(X(I)/2↑J)=0THEN180
175 F$(I)=F$(I)+"1":X(I)=X(I)-2↑J:GOTO185
180 F$(I)=F$(I)+"0"
185 NEXT:L$=L$+F$(I):RETURN
```

```
190 PRINTD;" ";
195 M=0:IFM$="0"THENRETURN
200 FORJ=LEN(M$)TO1STEP-1
205 IFMID$(M$,J,1)="1"THENM=M+2↑(LEN(M$)-J)
210 NEXTJ:RETURN
215 D=0:IFD$="0"THENRETURN
220 FORJ=1TOLEN(D$)
225 IFMID$(D$,J,1)="1"THEND=D+(1/2)↑J
230 NEXTJ:RETURN
235 N1$="STRINGA BIT CON AGGIUNTO BIT INIZIALE"
240 N2$="AL POSTO DEL BIT DI SEGNO"
245 O$="QUATTRO BYTE MANTISSA IN BINARIO"
250 P$="NUMERI FLOATING POINT"
255 C1$="NUMERO INTRODOTTO = "
260 C2$="QUATTRO BYTE MANTISSA IN MEMORIA"
265 C3$="STRINGA DI BIT INIZIALE"
270 C4$="NUMERO CALCOLATO = "
275 C5$="ESP. PER BASE 2 = "
280 C6$="VAL.BIN.PART.INT. SENZA SEGNO"
285 C7$="VAL.BIN.PART.DEC. SENZA SEGNO"
290 C8$="BYTE ESP. = "
295 Z$="0":FORK=1TO127:Z$=Z$+"0":NEXTK
300 RETURN
```

Il programma chiede un numero reale e stampa:
.il numero introdotto,
.i 4 byte della mantissa in decimale,
.i 4 byte della mantissa in binario,
.la stringa di bit dei 4 byte,
.la stringa di bit con aggiunto il bit 1 iniziale al posto del bit di segno,
.il valore decimale dell'esponente,
.l'esponente calcolato per la base 2 (ottenuto sottraendo 128 al valore precedente),
.il valore binario della parte intera senza segno,
.il valore binario della parte non intera,
.il numero ricalcolato.
In certi casi puoi trovare una differenza tra il numero introdotto e quello ricalcolato. La routine che ricalcola il numero, dopo aver aggiunto il bit 1 nella prima posizione a sinistra, si trova da 215 a 230.

Il nome della VARIABILE SINGOLA REALE sta nei primi due byte; troviamo nel primo il codice ASCII della prima lettera e nel secondo il codice ASCII del secondo carattere o zero.

Il programma VARINMEM ci consente di vedere la rappresentazione in memoria di qualunque tipo di variabile. Esso definisce le seguenti variabili:
.singole: M, N, Y, Z, A, K, B%, I, C$, J, Y$,
.con indice: D, E%, F$.
Servendosi dei puntatori vengono stampati i contenuti delle diverse zone di memoria dedicate alle variabili.

```
1 REM VARINMEM
10 M=0:N=0:Y=0:Z=0
15 DIMD(6),E%(5,4),F$(5,3,2)
20 A=0:FORK=1TO50:A=A+1:NEXTK
25 FORK=0TO6:D(K)=K*3:NEXTK
30 B%=0:FORI=1TO50STEP2:B%=B%+2:NEXTI
35 FORK=0TO5:FORI=0TO4
40 E%(K,I)=K*I+9:NEXTI,K
45 C$="FINE"
50 FORK=0TO5:FORJ=0TO3:FORI=0TO2
55 READY$:F$(K,J,I)=C$+Y$
60 NEXTI,J,K
65 Y=PEEK(45)+256*PEEK(46)
70 OPEN4,4:CMD4
75 PRINT:PRINT"VARIABILI":M=Y
80 PRINTM;"**"
85 FORI=0TO10:FORN=0TO6
90 Y=PEEK(45)+256*PEEK(46)
95 PRINTPEEK(M+N);"  ";:NEXTN:PRINT:PRINT
100 M=M+7:NEXTI
105 Y=PEEK(47)+256*PEEK(48)
110 Z=PEEK(49)+256*PEEK(50)
115 PRINT:PRINT"ARRAY":PRINTY;"**":K=Y
120 M=PEEK(K+2)+256*PEEK(K+3)
125 I=0:FORJ=KTOK+M-1
130 PRINTPEEK(J);" ";:I=I+1
135 IFI=7THENPRINT:I=0
140 NEXTJ:K=K+M:IFK=ZTHEN150
145 PRINT:GOTO120
150 PRINT:PRINT"FINE ARRAY: ";PEEK(Z)
155 PRINT:PRINT:PRINT"CORPO STRINGHE":PRINT
160 Y=PEEK(51)+256*PEEK(52)
165 Z=PEEK(55)+256*PEEK(56)
170 I=0:FORK=YTOY+50:PRINTK;"*";PEEK(K);
175 I=I+1:IFI=3THENPRINT:I=0
180 NEXTK
185 I=0:FORK=Z-50TOZ:PRINTK;"*";PEEK(K);
190 I=I+1:IFI=3THENPRINT:I=0
195 NEXTK
200 PRINT#4:CLOSE4:STOP
205 DATAAAA,BBB,CCC,DDD,EEE,FFF,GGG,III,LLL
210 DATAA,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T
215 DATAU,V,W,X,Y,Z,A1,B1,C1,D1,E1,F1,G1,H1,I1
220 DATAJ1,K1,L1,M1,N1,O1,P1,Q1,R1,S1,T1,U1,V1
225 DATAW1,X1,Y1,Z1,ABC,DEF,GHI,JKL,MNO,PQR,STU
230 DATAVWX,YZZ,PIPPO,PLUTO
```

Nella memoria destinata alle variabili del programma BASIC esistono tre zone distinte; esse sono:
.variabili singole, subito dopo il programma, puntatore in 45 e 46,
.variabili con indice, dopo le variabili singole, puntatore in 47 e 48,
.corpi delle stringhe, a partire dal fondo della memoria, per indirizzi decrescenti, puntatori in 51/ 52 e in 53/54.
Le stringhe sono rappresentate in due parti, la testata, con il nome, la lunghezza in caratteri, il puntatore al corpo della stringa e le altre indicazioni necessarie per le variabili stringa con indice, sta o nella prima o nella seconda zona, mentre il corpo della stringa, cioe' i caratteri, stanno nella terza zona. Quando il contenuto di una variabile stringa varia, si ha l'aggiornamento della testata e del corpo. Dal momento che esiste questa separazione in zone delle variabili, quando viene creata una variabile singola, le variabili con indice gia' esistenti vengono traslate in memoria per far posto alla nuova variabile.
Nel programma VARINMEM vengono create le variabili, viene letto con la funzione PEEK il contenuto e stampato.
La prima parte dei risultati ci mostra, dopo la parola VARIABILI, il valore del puntatore alle variabili, poi troviamo le 11 variabili singole del programma; ognuna occupa 7 byte. Troviamo ordinatamente:
M, nome 77 0, caratteristica 141, mantissa 33 56 0 0.
N, nome 78 0, caratteristica 130, mantissa 64 0 0 0.
Y, nome 89 0, caratteristica 141, mantissa 33 56 0 0.
Z, nome 90 0, caratteristica e mantissa presenti al momento della lettura;
A, nome 65 0, caratteristica e mantissa presenti al momento della lettura;
K, nome 75 0, caratteristica 131, mantissa 64 0 0 0.
B%, nome 194 128, valore intero 0 50;
I, nome 73 0, caratteristica 131, mantissa 96 0 0 0.
C$, nome 67 128, cioe' il codice ASCII del primo carattere del nome e il codice ASCII del secondo carattere del nome + 128 a significare il suffisso $; il terzo byte contiene il contatore dei caratteri, che puo' essere al massimo 255 e qui e' 4, infatti C$ contiene FINE, che le viene assegnato alla linea 45 del programma. Il quarto e quinto byte contengono il puntatore al corpo della stringa nel formato byte basso e byte alto. Nella zona corpo delle stringhe troviamo la parola FINE.
J, nome 74 0, caratteristica 131, mantissa 0 0 0 0.
Y$, nome 89 128, numero caratteri 5, puntatore a 60*256+159=15519. Da 15519 a 15523 si trova la parola PLUTO, ultima letta in Y$.

Andando a cercare nella terza zona dei risultati, quella denominata CORPO STRINGHE, troviamo la parola FINE seguita da due byte che contengono il puntatore al byte che contiene il numero di caratteri della stringa nella sua testata. Analogamente per la stringa Y$, troviamo la parola PLUTO, seguita dal puntatore al byte che da' la lunghezza della stringa nella sua testata.

Passando alla seconda zona dei risultati troviamo le variabili con indice. Dopo la parola ARRAY (denominazione inglese delle variabili con indice, dette anche MATRICI), seguita dall'indirizzo di inizio delle stesse.
Seguono ordinatamente:
D(6), cioe' la variabile numerica D di 7 elementi reali. Nei primi 2 byte compare il nome, 68 0; seguono due byte, byte basso e byte alto, che danno il numero totale dei byte occupati, in questo caso 42. Segue un byte con il numero delle dimensioni, 1 in questo caso. Si hanno poi due byte, byte alto e byte basso, contenenti il numero di elementi dell'unica dimensione, 7 in questo caso. Seguono tanti gruppi di 5 byte quanti sono gli elementi, contenenti i numeri reali nel formato caratteristica e mantissa.

E%(5,4), cioe' la variabile numerica intera con indice E% di 30 elementi (6*5). Vediamo 2 byte per il nome, 197 128; seguono i due byte contatori della memoria occupata, 69 byte qui. Segue il byte con il numero delle dimensioni, 2, il numero di elementi dell'ultima dimensione, 0 5, e infine il numero di elementi della prima dimensione, 0 6. Gli elementi che seguono occupano ciascuno 2 byte, dato che si tratta di numeri interi. Come vedi nel caso delle variabili con indice intere si ha un risparmio di memoria rispetto alle singole.
F$(5,3,2), cioe' la variabile stringa con indici F$, di 72 elementi. Per la testata valgono le stesse regole valide per le altre variabili con indice. Per gli elementi sono usati 3 byte; il primo dice quanto e' lungo l'elemento e gli ultimi due sono il puntatore al corpo della stringa.
Nella zona ARRAY, la descrizione di D(6) inizia alla prima riga e occupa 6 righe. La descrizione di E%(5,4) inizia alla settima riga e termina alla sedicesima riga. La descrizione di F$(5,3,) inizia alla diciassettesima riga (70 128 227 0 3 0 3); se conti 11 numeri, cioe' passi al quinto numero della riga seguente trovi: 7 226 63, cioe' il primo elemento F$(0,0,0), che contiene la parola FINEAAA.

Nella zona CORPO STRINGHE, se ti interessa, con un po' di pazienza puoi ritrovare i dati che desideri. L'unico problema e' calcolare bene i valori dati da due byte, ricordando quando il byte alto precede quello basso e quando no.

APPENDICE F

MAPPA DELLA MEMORIA

Label	ind. esadec.	ind. decimale	Descrizione
PDIR	$0000	0	7501 DATA DIRECTION REGISTER: direzione dei dati nella porta I/O nella CPU che si trova all'indirizzo 1. 0 = INPUT 1=OUTPUT, per ciascuno degli 8 bit
PORT	$0001	1	7501 DATA REGISTER: Registro dove leggere o scrivere lo stato dei piedini di I/O della CPU
SRCHTK	$0002	2	Temporanea per il BASIC
ZPVEC1	$0003-$0004	3-4	Temp (renumber)
ZPVEC2	$0005-$0006	5-6	Temp (renumber)
CHARAC	$0007	7	Temporanea per il BASIC
ENDCHR	$0008	8	Flag: cerca le virgolette a fine stringa
TRMPOS	$0009	9	Colonna dello schermo dall'ultimo TAB
VERCK	$000A	10	Flag: 0 = load; 1 = verify
COUNT	$000B	11	Punt. del buffer di Input/# di subscripts
DIMFLG	$000C	12	Flag: Dimensione di default per il vettore
VALTYP	$000D	13	Tipo di dato: $FF = stringa; $00 = numero
INTFLG	$000E	14	Tipo di dato: $80 = intero; $00 = floating
DORES	$000F	15	Flag: DATA scan/List quote/garbage coll
SUBFLG	$0010	16	Flag: subscript ref / user function call
INPFLG	$0011	17	Flag: $00 = INPUT; $40 = GET; $98 = READ
TANSGN	$0012	18	Flag: segno TAN / risultato del confronto

CHANNL	$0013	19	Flag: INPUT prompt
LINNUM	$0014-$0015	20-21	Temp: valore numero intero
TEMPPT	$0016	22	Puntatore: stack temporaneo per stringhe
LASTPT	$0017-$0018	23-24	Indirizzo ultima stringa temporanea
TEMPST	$0019-$0021	25-33	Stack per stringhe temporanee
INDEX1	$0022-$0023	34-35	Area per puntatori di utilita'
INDEX2	$0024-$0025	36-37	Area per puntatori di utilita'
RESHO	$0026	38	
RESMOH	$0027	39	
RESMO	$0028	40	
RESLO	$0029	41	
	$002A	42	
TXTTAB	$002B-$002C	43-44	Puntatore: inizio area programma BASIC
VARTAB	$002D-$002E	45-46	Punt: Fine programma BASIC+1 e inizio var.
ARYTAB	$002F-$0030	47-48	Punt: Fine variabili+1 e inizio vettori
STREND	$0031-$0032	49-50	Punt: Fine vettori+1
FRETOP	$0033-$0034	51-52	Punt: Fondo della zona stringhe
FRESPC	$0035-$0036	53-54	Puntatore di utilita' per le stringhe
MEMSIZ	$0037-$0038	55-56	Punt: Cima memoria riservata al BASIC+1
CURLIN	$0039-$003A	57-58	Numero della linea BASIC corrente
TXTPTR	$003B-$003C	59-60	
FNDPNT	$003D-$003E	61-62	
DATLIN	$003F-$0040	63-64	Numero della linea DATA corrente
DATPTR	$0041-$0042	65-66	Puntatore: indirizzo voce DATA corrente
INPPTR	$0043-$0044	67-68	Vettore: Routine INPUT
VARNAM	$0045-$0046	69-70	Nome della variabile BASIC corrente
VARPNT	$0047-$0048	71-72	Punt: dati della variabile BASIC corrente
FORPNT	$0049-$004A	73-74	Punt: variabile indice per FOR/NEXT
OPPTR	$004B-$004C	75-76	
OPMASK	$004D	77	

DEFPNT	$004E-$004F	78-79	
DSCPNT	$0050-$0051	80-81	
	$0052	82	
HELPER	$0053	83	
JMPER	$0054	84	
SIZE	$0055	85	
OLDOV	$0056	86	
TMPF1	$0057	87	
HIGHDS	$0058-$0059	88-89	
HIGHTR	$005A-$005B	90-91	
	$005C	92	
LOWDS	$005D-$005E	93-94	
LOWTR	$005F	95	
EXPSGN	$0060	96	
FACEXP	$0061	97	Esponente accumulatore virgola mobile 1
FACHO	$0062-$0065	98-101	Mantissa accumulatore virgola mobile 1
FACSGN	$0066	102	Segno accumulatore virgola mobile 1
SGNFLG	$0067	103	Puntatore: costante per calcolo in serie
BITS	$0068	104	Overflow accumulatore virgola mobile 1
ARGEXP	$0069	105	Esponente accumulatore virgola mobile 2
ARGHO	$006A-$006D	106-109	Mantissa accumulatore virgola mobile 2
ARGSGN	$006E	110	Segno accumulatore virgola mobile 2
ARISGN	$006F	111	Risultato confronto segni: accumul.1=acc.2
FACOV	$0070	112	Arrotondam. accumulatore virgola mobile 1
FBUFPT	$0071-$0072	113-114	Puntatore: Buffer cassetta
AUTINC	$0073-$0074	115-116	Valore di incremento per AUTO 0=escluso
MVDFLG	$0075	117	Flag: 10 K alta risoluzione allocati
KEYNUM	$0076	118	
KEYSIZ	$0077	119	
SYNTMP	$0078	120	

DSDESC	$0079-$007B	121-123	Descrittore di DS$
TOS	$007C-$007D	124-125	Cima dello stack del BASIC
TMPTON	$007E-$007F	126-127	Temporanee per la musica (tono e volume)
VOICNO	$0080	128	
RUNMOD	$0081	129	
POINT	$0082	130	
GRAPHM	$0083	131	Modo grafico corrente
COLSEL	$0084	132	Colore corrente selezionato
MC1	$0085	133	Colore multicolore 1
FG	$0086	134	Colore FOREGROUND
SCXMAX	$0087	135	Numero massimo di colonne
SCYMAX	$0088	136	Numero massimo di righe
LTFLAG	$0089	137	Flag: dipingi a sinistra
RTFLAG	$008A	138	Flag: dipingi a destra
STOPNB	$008B	139	Smette di dipingere se non sfondo
GRAPNT	$008C-$008D	140-141	
VTEMP1	$008E	142	
VTEMP2	$008F	143	
STATUS	$0090	144	Parola di stato del KERNAL (ST)
STKEY	$0091	145	Flag: Tasto STOP/tasto Reverse
SPVERR	$0092	146	Temporanea
VERFCK	$0093	147	Flag: 0=load; 1=verify
C3PO	$0094	148	Flag: Serial bus-Carattere in uscita
BSOUR	$0095	149	Carattere bufferizzato per serial bus
XSAV	$0096	150	Temporaneo
LDTND	$0097	151	Numero files aperti/indice a tabella file
DFLTN	$0098	152	Unita' di input di default (0)
DFLTO	$0099	153	Unita' di output di default (3)
MSGFLG	$009A	154	Flag: $80 = modo diretto; $00=programma
SAL	$009B	155	Errore passo 1 su nastro

SAH	$009C	156	Errore passo 2 su nastro
EAL	$009D	157	
EAH	$009E	158	
T1	$009F-$00A0	159-160	Area dati temporanea
T2	$00A1-$00A2	161-162	Area dati temporanea
TIME	$00A3-$00A5	163-165	Orologio in tempo reale in 1/60 sec.
R2D2	$00A6	166	Usata dal Serial Bus
TPBYTE	$00A7	167	Byte da essere scritto/letto su/da nastro
BSOUR1	$00A8	168	Temporanea usata dalla routine seriale
FPVERR	$00A9	169	
DCOUNT	$00AA	170	
FNLEN	$00AB	171	Lunghezza del nome del file corrente
LA	$00AC	172	Numero del file logico corrente
SA	$00AD	173	Indirizzo secondario corrente
FA	$00AE	174	Numero di unita' corrente
FNADR	$00AF-$00B0	175-176	Punt.: nome del file corrente
ERRSUM	$00B1	177	
STAL	$00B2	178	Indirizzo di inizio I/O
STAH	$00B3	179	
MEMUSS	$00B4-$00B5	180-181	Inizio della RAM da caricare
TAPEBS	$00B6-$00B7	182-183	
TMP2	$00B8-$00B9	184-185	
WRBASE	$00BA-$00BB	186-187	Puntatore ai dati per scrittura su nastro
IMPAR	$00BC-$00BD	188-189	
FETPTR	$00BE-$00BF	190-191	Punt: byte da indirizzare nell'indirizzamento a banchi
SEDSAL	$00C0-$00C1	192-193	Temporaneo per lo Scrolling
RVS	$00C2	194	Flag di reverse on
INDX	$00C3	195	
LSXP	$00C4	196	Posizione X all'inizio

LSTP	$00C5	197	
SFDX	$00C6	198	Tasto premuto
CRSW	$00C7	199	Flag: INPUT o GET dalla tastiera
PNT	$00C8-$00C9	200-201	Punt: indirizzo linea corrente di schermo
PNTR	$00CA	202	Colonna del cursore nella linea corrente
QTSW	$00CB	203	Flag: Editor in modo virgolette (0=NO)
SEDT1	$00CC	204	Uso temporaneo per editor
TBLX	$00CD	205	Numero di linea fisica del cursore
DATAX	$00CE	206	Area dati temporanea
INSRT	$00CF	207	Flag: modo Insert, >0=numero di insert
FREE	$00D0-$00E8	208-232	Area libera per l'utente
CIRSEG	$00E9	233	Tabella di collegamento linee schermo
USER	$00EA-$00EB	234-235	
KEYTAB	$00EC-$00ED	236-237	Vettore tabella scansione tastiera
TMPKEY	$00EE	238	
NDX	$00EF	239	Indice alla coda della tastiera
STPFLG	$00F0	240	Flag: pausa
TO	$00F1-$00F2	241-242	Locazione in pagina 0 usata dal MONITOR
CHRPTR	$00F3	243	
BUFEND	$00F4	244	
CHKSUM	$00F5	245	Temporanea per il controllo della parita'
LENGHT	$00F6	246	
PASS	$00F7	247	
TYPE	$00F8	248	Tipo di blocco
USEKDY	$00F9	249	
XSTOP	$00FA	250	Salva registro X per controllo rapido STOP
CURBNK	$00FB	251	Configurazione del banco corrente
XON	$00FC	252	Codice carattere XON (RS 232)
XOFF	$00FD	253	Codice carattere XOFF (RS 232)
SEDT2	$00FE	254	Usata temporaneamente dall'editor

LOFBUF	$00FF	255	
SYSSTK	$0100-$01FF	256-511	Stack della CPU
BUF	$0200-$0258	512-600	Buffer di input BASIC e MONITOR
	$0259-$02AC	601-684	Area usata dal BASIC per comandi DOS
	$02AD-$02F1	685-753	Area usata dal BASIC per comandi grafici
ADRAY1	$02F2-$02F3	754-755	Punt: conversione FLOATING-INTERO
ADRAY2	$02F4-$02F5	756-757	Punt: conversione INTERO-FLOATING
BNKVEC	$02FE-$02FF	766-767	Vettore per l'uso di funzioni su cartridge
IERROR	$0300-$0301	768-769	Vettore errore (codice in reg. X)
IMAIN	$0302-$0303	770-771	Vettore esecuzione istruzione BASIC
ICRNCH	$0303-$0304	772-773	Vett. tokenizzaz. (compatta parole) chiave del BASIC)
IQPLOP	$0306-$0307	774-775	Vettore routine LIST
IGONE	$0308-$0309	776-777	Vettore routine analisi prossimo carattere del programma BASIC
IEVAL	$030A-$030B	778-779	Vettore per la valutazione dei simboli
IESCLK	$030C-$030D	780-781	
IESCPR	$030E-$030F		
IESCEX	$0310-$0311		
ITIME	$0312-$0313		
CINV	$0314-$0315	788-789	Vettore di INTERRUPT
CBINV	$0316-$0317	790-791	Vettore BRK
IOPEN	$0318-$0319		VETTORI DEL SISTEMA OPERATIVO
ICLOSE	$031A-$031B		
ICHKIN	$031C-$031D		
ICKOUT	$031E-$031F		
ICLRCH	$0320-$0321		
IBASIN	$0322-$0323		
IBSOUT	$0324-$0325		
ISTOP	$0326-$0327		

IGETIN	$0328-$0329		
ICLALL	$032A-$032B		
USRCMD	$032C-$032D		
ILOAD	$032E-$032F		
ISAVE	$0330-$0331		
TAPBUF	$0333-$03F2	819-1010	Buffer per il registratore a cassetta
WRLEN	$03F3-$03F4	1011-1012	Lunghezza dei dati da scrivere su nastro
RDCNT	$03F5-$03F6	1013-1014	Lunghezza dei dati da leggere da nastro
	$03F7-$0472	1015-1138	Coda ingresso rs 232
CHRGET	$0473-$0478	1139-1144	Routine di lettura caratteri da memoria usata dal BASIC per ricevere un carattere alla volta
CHRGOT	$0479-$0484	1145-1156	Seguito di CHRGET, ma entrando da questo punto ritorna l'ultimo carattere letto
USRPOK	$0500-$0503	1280-1282	Vettore di salto funzione USR
LAT	$0509-$0512	1189-1198	Tabella dei numeri di file logico
FAT	$0513-$051C	1299-1308	Tabella indirizzi primari
SAT	$051D-$0526	1309-1318	Tabella indirizzi secondari
KEYD	$0527-$0530	1319-1328	Buffer della tastiera
	$0530-$07F1	1329-2033	Usata per RS 232
SAREG	$07F2	2034	Registri per il comando SYS: accumulatore
SXREG	$07F3	2035	Registro X
SYREG	$07F4	2036	Registro Y
SPREG	$07F5	2037	Registro dei FLAG
LSEM	$07FC	2044	Semaforo per arresto motore cassetta
TEDATR	$0800-0BFF	2048-3071	Mappa degli attributi per il video
TEDSCN	$0C00-0FFF	3072-4095	Mappa dei caratteri nel video

APPENDICE G

REGISTRI DEL TED

Segue una tabella che riporta gli indirizzi esadecimali e decimali dei registri del TED e la funzione dei bit di ogni registro.

INDIRIZZO HEX	INDIRIZZO DEC	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
$FF00	65280	TIMER 1 low							
$FF01	65281	TIMER 1 high							
$FF02	65282	TIMER 2 low							
$FF03	65283	TIMER 2 high							
$FF04	65284	TIMER 3 low							
$FF05	65285	TIMER 3 high							
$FF06	65286	-	1 = sfondo programmabile	1 = modo grafico	0 = annullamento dello schermo	0 = 24 righe 1 = 25 righe	posizione delle scritte per scorrimento verticale		
$FF07	65287	1 = disabilita reverse	0 = PAL 1 = NTSC	-	1 = multicolore	0 = 38 colonne 1 = 40 colonne	posizione delle scritte per scorrimento orizzontale		
$FF08	65288	LATCH DELLA TASTIERA							
$FF09	65289	INTERRUPT FLAG							
		interrupt flag	timer 3	-	timer 2	timer 1	light pen	raster register	-
$FF0A	65290	ABILITAZIONE DEGLI INTERRUPT							
		-	timer 3	-	timer 2	timer 1	light pen	raster register	vedi prossimo
$FF0B	65291	REGISTRO DI LINEA (scrittura per interrupt) bit 7-0 (il bit 8 e' il bit 0 del registro di abilitazione degli interrupt)							
$FF0C	65292	-	-	-	-	-	-	bit 9 della posizione del cursore	bit 8 della posizione del cursore
$FF0D	65293	POSIZIONE DEL CURSORE (bit 7-0)							
$FF0E	65294	FREQUENZA DELLA VOCE 1 (bit 7-0)							
$FF0F	65295	FREQUENZA DELLA VOCE 2 (bit 7-0)							
$FF10	65296	-	-	-	-	-	-	bit 9 frequenza voce 2	bit 8 frequenza voce 2
$FF11	65297	1 : non suona	1 : selezione rumore (voce 2)	1 : selezione voce 2	1 : selezione voce 1	VOLUME			
$FF12	65298	-	-	bit 15 indirizzo mappa grafica	bit 14 indirizzo mappa grafica	bit 13 indirizzo mappa grafica	0=TED legge RAM 1=TED legge ROM	bit 9 frequenza voce 1	bit 8 frequenza voce 1
$FF13	65299	bit 15 indirizzo descr. caratteri	bit 14 indirizzo descr. caratteri	bit 13 indirizzo descr. caratteri	bit 12 indirizzo descr. caratteri	bit 11 indirizzo descr. caratteri	bit 10 indirizzo descr. caratteri	selezione clock: 1=.89, 0=1.78MHz	-
$FF14	65300	bit 15 indirizzo mappe video attr	bit 14 indirizzo mappe video attr	bit 13 indirizzo mappe video attr	bit 12 indirizzo mappe video attr	bit 11 indirizzo mappe video attr	bit 10 indirizzo mappe video attr	-	-
$FF15	65301	COLORE DELLO SCHERMO							
$FF16	65302	COLORE DI SFONDO 1							
$FF17	65303	COLORE DI SFONDO 2							
$FF18	65304	COLORE DI SFONDO 3							
$FF19	65305	COLORE DEL BORDO							
$FF1A	65306	-							
$FF1B	65307	-							
$FF1C	65308	-	-	-	-	-	-	-	bit 8 registro linea (lettura)
$FF1D	65309	REGISTRO DI LINEA (lettura) bit 7-0							
$FF1E	65310	POSIZIONE ORIZZONTALE DEL PENNELLO DEL MONITOR							
$FF3E	65342	Scrivendo un qualunque dato in questo registro la CPU seleziona la ROM							
$FF3F	65343	Scrivendo un qualunque dato in questo registro la CPU seleziona la RAM							

VALORE DELLE NOTE

Questa appendice contiene la lista delle 72 note che formano 6 ottave, il valore che bisogna usare come secondo parametro nell'istruzione SOUND per ottenere la nota, e la frequenza della nota espressa in Hertz.

NOTA	VALORE	FREQUENZA
LA	7	109.97
LA#	64	116.50
SI	118	123.44
DO	169	130.81
DO#	217	138.59
RE	262	146.77
RE#	305	155.55
MI	345	164.71
FA	383	174.48
FA#	419	184.86
SOL	453	195.87
SOL#	485	207.50
LA	516	220.16
LA#	544	233.00
SI	571	246.89
DO	597	261.92
DO#	621	277.52
RE	643	293.54
RE#	665	311.53
MI	685	329.91
FA	704	349.50
FA#	722	370.33
SOL	739	392.42
SOL#	755	415.76

NOTA	VALORE	FREQUENZA
LA	770	440.32
LA#	784	466.00
SI	798	494.87
DO	810	522.62
DO#	822	553.67
RE	834	588.63
RE#	844	621.34
MI	854	657.89
FA	864	699.00
FA#	873	740.67
SOL	881	782.10
SOL#	889	828.45
LA	897	880.63
LA#	904	932.00
SI	911	989.74
DO	917	1045.24
DO#	923	1107.33
RE	929	1177.27
RE#	934	1242.67
MI	939	1315.77
FA	944	1398.01
FA#	948	1471.58
SOL	953	1575.22
SOL#	957	1669.26

NOTA	VALORE	FREQUENZA
LA	960	1747.51
LA#	964	1864.01
SI	967	1962.11
DO	971	2110.20
DO#	974	2236.81
RE	976	2330.01
RE#	979	2485.34
MI	982	2662.87
FA	984	2796.01
FA#	986	2943.17
SOL	988	3106.68
SOL#	990	3289.43
LA	992	3495.01
LA#	994	3728.02
SI	996	3994.30
DO	997	4142.24
DO#	999	4473.62
RE	1000	4660.02
RE#	1002	5083.66
MI	1003	5325.74
FA	1004	5592.02
FA#	1005	5886.34
SOL	1006	6213.36
SOL#	1007	6578.85

FUNZIONI MATEMATICHE DERIVATE

Questa appendice mostra come calcolare le funzioni che non sono direttamente richiamabili dal BASIC 3.5.

FUNZIONE	EQUIVALENTE BASIC
sec(X)	1/COS(X)
cosec(X)	1/SIN(X)
cotan(X)	1/TAN(X)
arcsin(X)	ATN(X/SQR(-X*X+1))
arccos(X)	ATN(X/SQR(-X*X+1))+π/2
arcsec(X)	ATN(X/SQR(X*X-1))
arccosec(X)	ATN(X/SQR(X*X-1))+ (SGN(X)-1)*π/2
arccotg(X)	ATN(X)+π/2
sinh(X)	(EXP(X)-EXP(-X))/2
cosh(X)	(EXP(X)+EXP(-X))/2
tgh(X)	EXP(-X)/(EXP(X)+ EXP(-X))*2+1
sech(X)	2/(EXP(X)+EXP(-X))
cosech(X)	2/(EXP(X)-EXP(-X))
cotgh(X)	EXP(-X)/(EXP(X)- EXP(-X))*2+1

FUNZIONE	EQUIVALENTE BASIC
arcsinh(X)	LOG(X+SQR(X*X+1))
arccosh(X)	LOG(X+SQR(X*X-1))
arctgh(X)	LOG((1+X)/(1-X))/2
arcsech(X)	LOG(SQR(-X*X+1)+1/X)
arccosch(X)	LOG(SGN(X)*SQR(X*X+ 1/X))
arccotgh(X)	LOG((X+1)/(X-1))/2

CONVERSIONI DEC./ESADEC./BIN.

Nella tabella che segue sono riportati i numeri decimali da 0 a 255 e i loro corrispondenti esadecimali.

DEC.	HEX.	DEC.	HEX.	DEC.	HEX.	DEC.	HEX.
0	00	32	20	64	40	96	60
1	01	33	21	65	41	97	61
2	02	34	22	66	42	98	62
3	03	35	23	67	43	99	63
4	04	36	24	68	44	100	64
5	05	37	25	69	45	101	65
6	06	38	26	70	46	102	66
7	07	39	27	71	47	103	67
8	08	40	28	72	48	104	68
9	09	41	29	73	49	105	69
10	0A	42	2A	74	4A	106	6A
11	0B	43	2B	75	4B	107	6B
12	0C	44	2C	76	4C	108	6C
13	0D	45	2D	77	4D	109	6D
14	0E	46	2E	78	4E	110	6E
15	0F	47	2F	79	4F	111	6F
16	10	48	30	80	50	112	70
17	11	49	31	81	51	113	71
18	12	50	32	82	52	114	72
19	13	51	33	83	53	115	73
20	14	52	34	84	54	116	74
21	15	53	35	85	55	117	75
22	16	54	36	86	56	118	76
23	17	55	37	87	57	119	77
24	18	56	38	88	58	120	78
25	19	57	39	89	59	121	79

TABELLA L.1 Conversione dec. esadec.

DEC.	HEX.	DEC.	HEX.	DEC.	HEX.	DEC.	HEX.
26	1A	58	3A	90	5A	122	7A
27	1B	59	3B	91	5B	123	7B
28	1C	60	3C	92	5C	124	7C
29	1D	61	3D	93	5D	125	7D
30	1E	62	3E	94	5E	126	7E
31	1F	63	3F	95	5F	127	7F

DEC.	HEX.	DEC.	HEX.	DEC.	HEX.	DEC.	HEX.
128	80	160	A0	192	C0	224	E0
129	81	161	A1	193	C1	225	E1
130	82	162	A2	194	C2	226	E2
131	83	163	A3	195	C3	227	E3
132	84	164	A4	196	C4	228	E4
133	85	165	A5	197	C5	229	E5
134	86	166	A6	198	C6	230	E6
135	87	167	A7	199	C7	231	E7
136	88	168	A8	200	C8	232	E8
137	89	169	A9	201	C9	233	E9
138	8A	170	AA	202	CA	234	EA
139	8B	171	AB	203	CB	235	EB
140	8C	172	AC	204	CC	236	EC
141	8D	173	AD	205	CD	237	ED
142	8E	174	AE	206	CE	238	EE
143	8F	175	AF	207	CF	239	EF
144	90	176	B0	208	D0	240	F0
145	91	177	B1	209	D1	241	F1
146	92	178	B2	210	D2	242	F2
147	93	179	B3	211	D3	243	F3
148	94	180	B4	212	D4	244	F4
149	95	181	B5	213	D5	245	F5
150	96	182	B6	214	D6	246	F6
151	97	183	B7	215	D7	247	F7
152	98	184	B8	216	D8	248	F8
153	99	185	B9	217	D9	249	F9
154	9A	186	BA	218	DA	250	FA
155	9B	187	BB	219	DB	251	FB
156	9C	188	BC	220	DC	252	FC
157	9D	189	BD	221	DD	253	FD
158	9E	190	BE	222	DE	254	FE
159	9F	191	BF	223	DF	255	FF

TABELLA L.1 Conversione dec. esadec. (continuazione)

Segue il programma CONV1 che stampa una tabellina contenente i numeri decimali da 0 a 15 e i corrispondenti numeri esadecimali e binari.

```
1 REM CONV1
6 OPEN4,4:CMD4
11 DATA0,0000,1,0001,2,0010,3,0011,4,0100
16 DATA5,0101,6,0110,7,0111,8,1000,9,1001
21 DATAA,1010,B,1011,C,1100,D,1101,E,1110,F,1111
26 PRINT"DECIMALE  ESADECIMALE  BINARIO"
31 PRINT
36 FORK=0TO15
41 READA$,B$:C$=STR$(K)
46 C$=RIGHT$("  "+C$,2)
51 PRINT"   ";C$;"          ";A$;"          ";B$
56 NEXTK
61 PRINT#4:CLOSE4
```

RISULTATI PROGRAMMA CONV1

DECIMALE	ESADECIMALE	BINARIO
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Nel programma CONV1 i numeri corrispondenti ai decimali, in esadecimale e binario, sono ottenuti dal file interno di dati generato dalle frasi DATA delle linee 11, 16 e 21, nelle quali sono riportati a coppie il numero in esadecimale e il corrispondente in binario.

Il COMMODORE PLUS-4 dispone di due funzioni in BASIC, la DEC e la HEX$, che consentono di operare conversioni da esadecimale in decimale e viceversa per numeri esadecimali di al massimo 4 cifre e per numeri decimali compresi tra 0 e 65535.
Il programma CONV2 che segue accetta come dato di ingresso una stringa esadecimale di al massimo 4 caratteri, e la converte nel numero decimale corrispondente.

```
1 REM CONV2
10 PRINT"♥SCRIVI UN NUMERO ESADECIMALE DI"
15 PRINT"AL MASSIMO 4 CIFRE"
20 INPUTN$
25 IFLEN(N$)>4THEN10
30 FORK=1TOLEN(N$)
35 A$=MID$(N$,K,1)
40 IFA$>="0"ANDA$<="9"ORA$>="A"ANDA$<="F"THEN50
45 GOTO10
50 NEXTK
60 PRINTN$;" IN DECIMALE = ";DEC(N$)
65 STOP
```

Nel programma CONV2 la stringa esadecimale da convertire viene controllata e accettata solo se e' di al massimo 4 caratteri e i caratteri sono cifre esadecimali, cioe' comprese negli intervalli 0-9 e A-F.

Il programma CONV3 che segue accetta in ingresso un numero decimale compreso tra 0 e 65535 e lo converte nella stringa esadecimale corrispondente.

```
1 REM CONV3
5 PRINT"♥SCRIVI UN NUMERO DECIMALE POSITIVO"
10 PRINT"COMPRESO TRA 0 E 65535"
15 INPUTN
20 IFN<0ORN>65535THEN5
25 PRINTN;" IN ESADECIMALE = ";HEX$(N)
30 STOP
```

Il programma CONV3 accetta in ingresso un numero positivo compreso tra 0 e 65535 e lo converte sempre in una stringa di 4 caratteri esadecimali.

Il programma CONV4 che segue consente di convertire un numero decimale, compreso tra 0 e 255, in binario; dopo averlo convertito e stampato, viene effettuata la controprova, cioe' il numero binario viene riconvertito in decimale e stampato. In questo caso le conversioni vengono operate effettuando dei calcoli.

```
1 REM CONV4
5 PRINT"♥SCRIVI UN NUMERO DECIMALE POSITIVO"
10 PRINT"COMPRESO TRA 0 E 255"
15 INPUTN
20 IFN<0ORN>255THEN5
25 X$="":A=N
30 FORK=7TO0STEP-1
35 IFN>=2↑KTHENX$=X$+"1":N=N-2↑K:GOTO45
40 X$=X$+"0"
45 NEXTK
50 PRINTA;" IN BINARIO = ";X$:PRINT
55 A=0:FORK=8TO1STEP-1
60 A$=MID$(X$,K,1)
65 IFA$="1"THENA=A+2↑(8-K)
70 NEXTK
75 PRINTX$;" IN DECIMALE = ";A
80 STOP
```

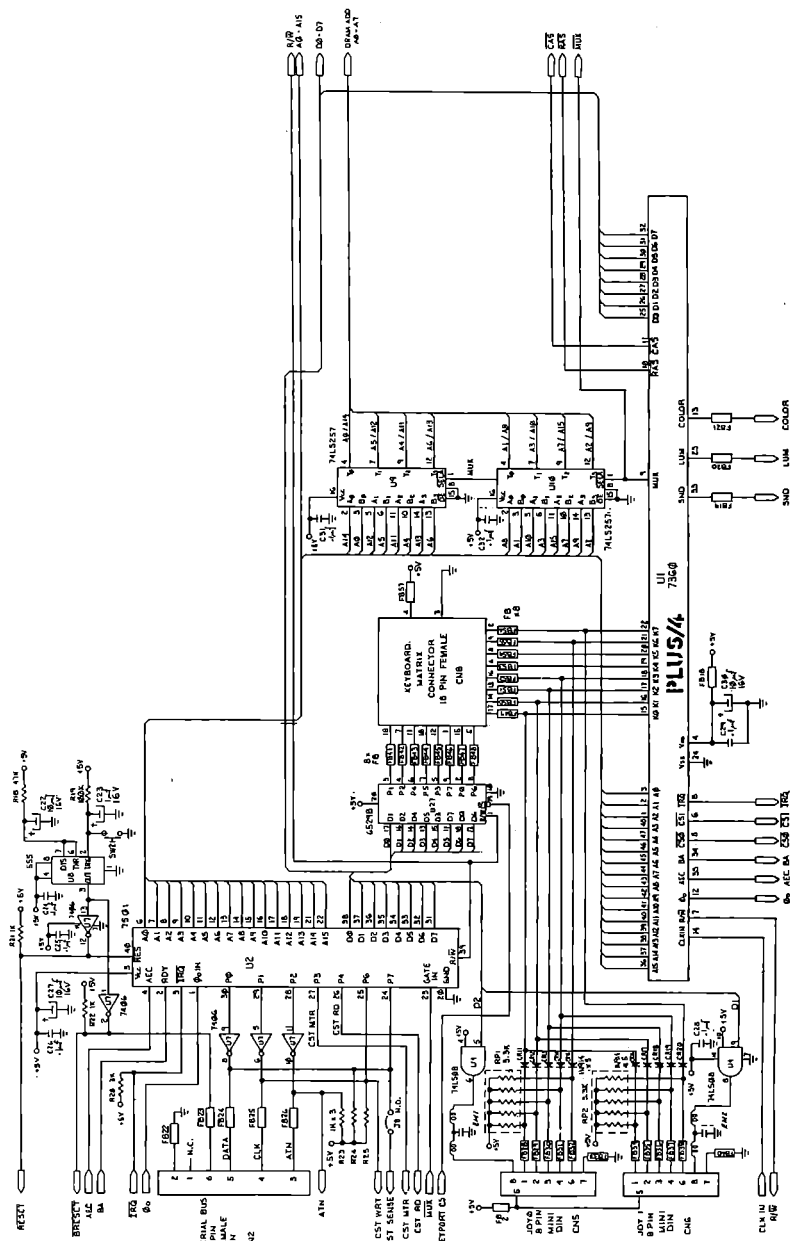
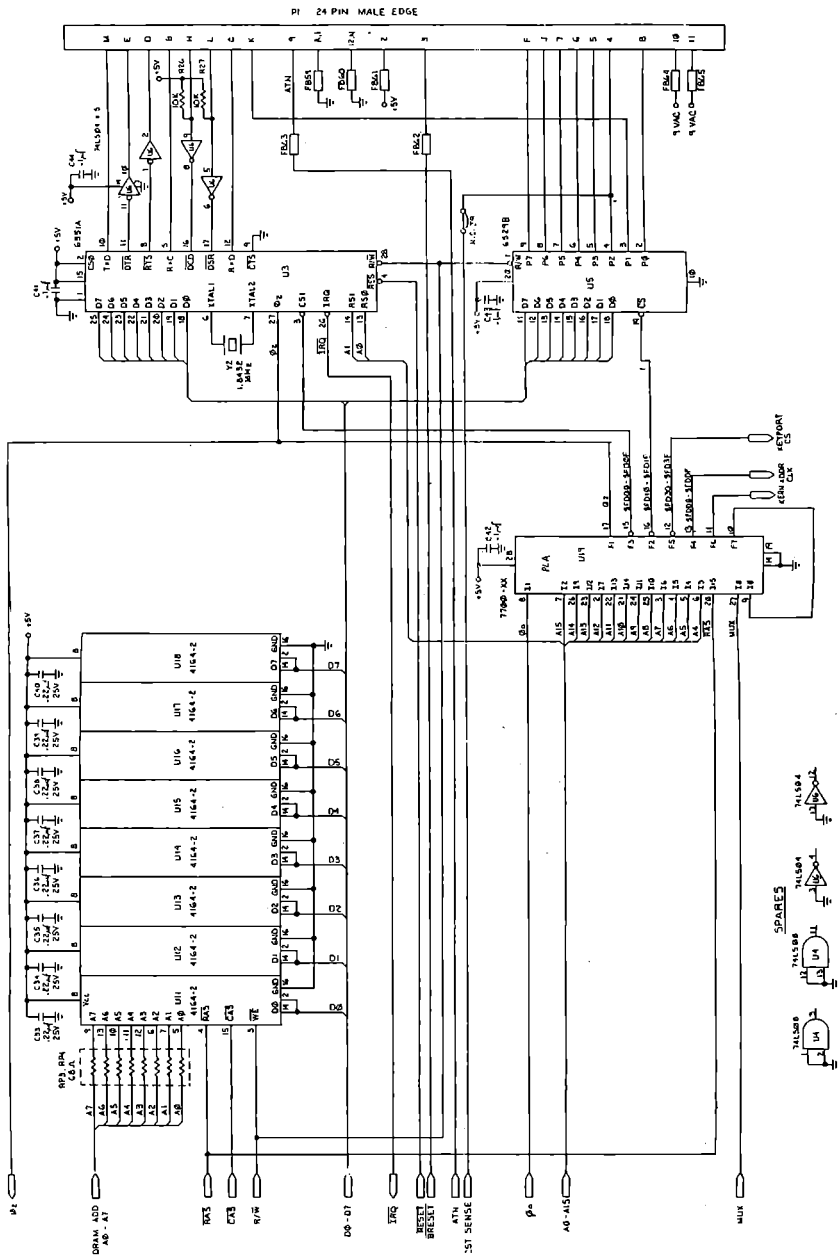
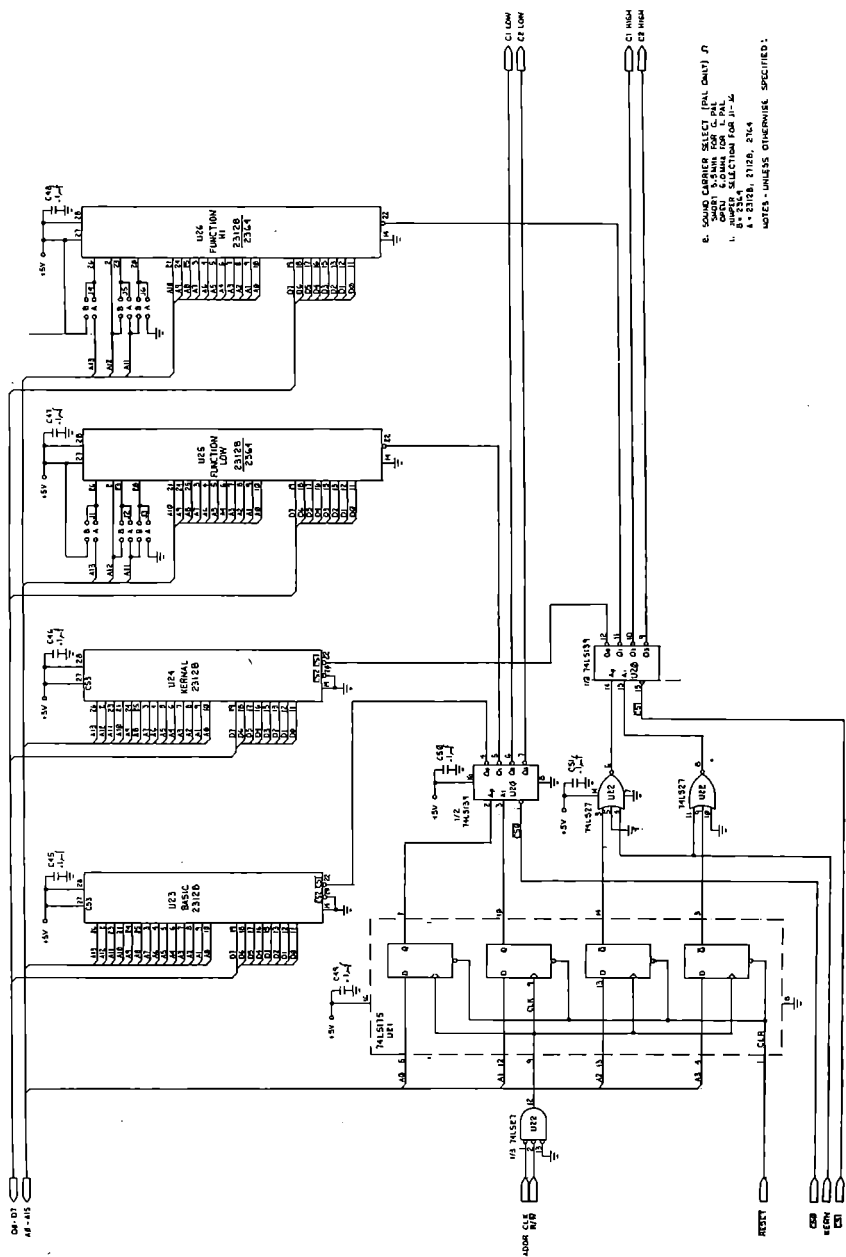
I 4 PROGRAMMI RESIDENTI IN ROM

Premendo il tasto F1 e quindi il tasto RETURN, il COMMODORE PLUS-4 entra in un ambiente diverso da quello BASIC. Qui puoi usare i 4 programmi memorizzati su ROM che caratterizzano questo calcolatore.

I 4 programmi sono:
WORD PROCESSOR: un programma che ti permette di scrivere, memorizzare e stampare testi con grande facilita'.
DATA BASE: programma che ti permette di memorizzare e gestire grandi quantita' di dati come quelli che compongono un'agenda telefonica, un listino prezzi o un inventario di magazzino.
SPREAD SHEET: programma che ti permette di memorizzare grandi quantita' di dati numerici in un tabellone e di mostrarti in modo immediato le relazioni che li legano.
GRAPHICS: permette di avere una visualizzazione grafica delle grandezze che sono contenute nel tabellone dello SPREAD SHEET.

Puoi trovare per ogni calcolatore programmi di questo tipo, ma la potenza che ti offre il pacco di programmi del PLUS-4 consiste nel fatto che i dati gestiti da uno dei programmi possono essere passati ad un altro; ad esempio puoi usare una lista di indirizzi memorizzati col DATA BASE per scrivere le intestazioni di una lettera fatta con WORD PROCESSOR o inserire un grafico che visualizza informazioni memorizzate nel tabellone elettronico in una lettera.
Il calcolatore viene venduto con dettagliati ed esaurienti manuali in italiano, per cui non abbiamo ritenuto opportuno approfondire ulteriormente l'argomento.

SCHEMA ELETTRICO

Segue lo schema elettrico del COMMODORE PLUS-4.

50 PIN FEMALE EDGE CONNECTOR CN4
AUDIO / VIDEO 8 PIN DIN CN7
CASSETTE 7 PIN MINI DIN CN3

P1 24 PIN MALE EDGE
6551A
6529B
PLA
SPARES

Questo libro vuole essere una guida alla conoscenza del Commodore Plus-4 e all'apprendimento della programmazione in linguaggio BASIC e in linguaggio Assembler.
Viene presentata l'implementazione del BASIC 3.5 disponibile sul Commodore Plus-4. Inoltre è illustrato il DOS residente sull'unità a floppy disk 1541 e viene insegnato l'uso completo della stampante grafica MPS-803.
Lo studio dell'Assembler viene sviluppato appoggiandosi alle possibilità offerte dal comando MONITOR del BASIC, ma viene affrontato anche il problema del linguaggio macchina della CPU 7501 e della sua struttura hardware.
Le spiegazioni teoriche sono sempre accompagnate da esempi commentati, e per questo il libro può essere utile anche ai principianti.
Particolare attenzione viene dedicata all'uso delle periferiche, compresa la gestione dei file su disco. La grafica è uno degli argomenti più approfonditi, collegandola alle possibilità del processore video.
Nei primi tre capitoli si ha una panoramica delle possibilità del Commodore Plus-4 e delle problematiche relative alla programmazione, affrontando gli argomenti in modo rigoroso, ma a livello elementare. Nei capitoli seguenti gli argomenti vengono approfonditi, ma si cerca di rendere le spiegazioni abbastanza semplici.
Il capitolo 4 tratta le periferiche: tastiera, video, cassetta e joistick. Il capitolo 5 presenta completamente la stampante grafica MPS-803 e riporta diversi programmi per ottenere la copia del video testo e del video grafico. Il capitolo 6 tratta i file su disco, riportando tutte le informazioni necessarie per usare bene il DOS; inoltre sono presenti programmi commentati per gestire archivi di dati di tipo sequenziale e di tipo random, random-user e relativo, anche con indice. Il capitolo 7 si sofferma sull'architettura del sistema ponendo l'accento sulla parte hardware e mettendo in grado anche gli inesperti di comprendere come è costruito il calcolatore. Il capitolo 8 tratta del linguaggio macchina, dell'Assembler, dell'uso completo del comando MONITOR e del software residente. Il capitolo 9 approfondisce l'argomento della grafica, utilizzando anche il linguaggio macchina. Nel capitolo 10 sono presenti alcuni programmi commentati.
Completano il volume le appendici, che riportano: la scheda completa del BASIC 3.5, le istruzioni Assembler, i codici e i numeri del sistema, i messaggi d'errore, la mappa e l'utilizzo della memoria, l'elenco dei registri di I/O, il valore delle note, le funzioni matematiche derivate, le conversioni tra i sistemi di numerazione, una breve sintesi dei 4 programmi residenti in ROM e lo schema elettrico del calcolatore.

L.30.000 Cod. CC256 ISBN 88-7056-335-9